MASTER IN COMPUTER SCIENCE

A contribution to the analysis of the transport protocols

Schaeck, Sébastien

Award date:
1990

Awarding institution:
University of Namur

Facultés Universitaires
Notre-Dame de la Paix
NAMUR

Institut d'Informatique

A contribution to the analysis of the transport protocols.

Sébastien Schaeck.

Promoteur:
Philippe van Bastelaer.

Mémoire présenté en vue
de l'obtension du titre
de Licencié et Maître
en Informatique.

Année académique 1989-1990.

Abstract.

Most transport protocols that are currently used have been designed when the networking technology was still very poor. Their only requirement was to stream data between two computing equipments connected to a network of slow links.

Since that period, the networking technology has radically evolved, mainly at the speed point of view. But the achievable end-to-end throughputs are often an order of magnitude lower than the bandwidth these networks provide: the popular transport protocols can be blamed for a severe lack of performance.

The scope of this thesis is to analyze the transport protocols at the performance point of view. The first step will consist of the explanation of this lack of end-to-end throughput. And the second one will be the presentation of the actually proposed solutions, with an analysis of their merits.

La plupart des protocoles de transport qui sont utilisés aujourd'hui ont été créés lorsque la technologie des réseaux informatiques était encore fort primitive. La seule fonction qui leur était attribuée était de transférer un flux de données entre deux équipements informatiques connectés à un réseau de lignes à basse vitesse.

Depuis ce temps, la technologie des réseaux informatiques a beaucoup évolué, surtout au niveau des vitesses de transmission. Mais les vitesses de transmission de bout en bout restent bien inférieures à la bande passante de ces réseaux: un important manque de performances peut être reproché aux protocoles de transport couramment utilisés.

Le propos de ce mémoire est d'analyser les protocoles de transport, du point de vue de leurs performances. Une première étape va consister à donner une explication à ce manque de vitesse de transmission de bout en bout. La seconde sera la présentation des solutions actuellement proposées, et l'analyse de leurs qualités respectives.

Acknowledgements.

First of all, I wish to express sincere thanks to a few people who had a non negligible impact on the realization of this thesis:

- Patrick Geurts, from BIM s.a./n.v., for his helpful guidance about my study of the transport protocols,
- Philippe van Bastelaer, professor at the university of Namur, Belgium, the promoter of this work, for the complete freedom of action he gave me, as well as for his beneficial comments about a preceding draft of this document,
- Bernard Detrembleur, his main assistant, for the attention he paid to the last draft of this thesis, and his advices to improve it.

Table of content.

Introduction.

Background.

Most transport protocols that are nowadays used have been designed a very long time ago. TCP, for example, which is the standard transport protocol of the Department of Defense, was defined in 1975, already fifteen years ago. It is no surprise these protocols were defined according to the paradigms, the available means and the needs of that moment.

The general model of computer systems consisted of a central mainframe interacting with its users via text terminals connected to a wide area network (WAN). This mainframe was very independent from its peers, only interacting with them through this WAN for sporadic file transfers.

The purpose of these wide area networks is to connect a large amount of computing equipments distributed over a large geographical region. The technology they involved at that time was in fact very poor. The links they used were characterized by:

- a relatively low reliability,
- a very low capacity, with a bandwidth of 9600 bits per second, in general, but never exceeding 64 kilobits per second.

So the needs in the matter of transport protocols were not very constraining. Their only requirement was to allow a reliable transmission of a stream of bytes between two equipments. This functionality was sufficient to connect text terminals to central hosts, and to ensure background file transfers.

But since that period, the technology has evolved. This evolution conducted to the formulation of new paradigms for the computer science. And these new paradigms induced new requirements for the transport protocols.

The main evolution of the technology is the conception of the local area networks (LANs). The purpose of these LANs is to connect a median amount of computer equipments, which are geographically close one to the others. Their main characteristic is the bandwidth they offer: up to ten megabits per second. The technology of the WANs has also evolved, with the use of fast links, with a capacity reaching two megabits per second. Finally, massive developments are realized on optic fibres: these are expected to allow a hundred megabits per second networking capacity in a near future, and later a gigabit per second.

This strong evolution of the networking capacity had an impact on the paradigms of the computer science. First, not only text terminals can be connected to a host, but also bitmaps ones. These terminals need to exchange a really more important amount of information with their host. Second, distributed computer systems can be designed. With this model, the hosts does no more sporadically interact: they do all the time, for the purpose of exchanging massive quantities of information, with real-time constraints.

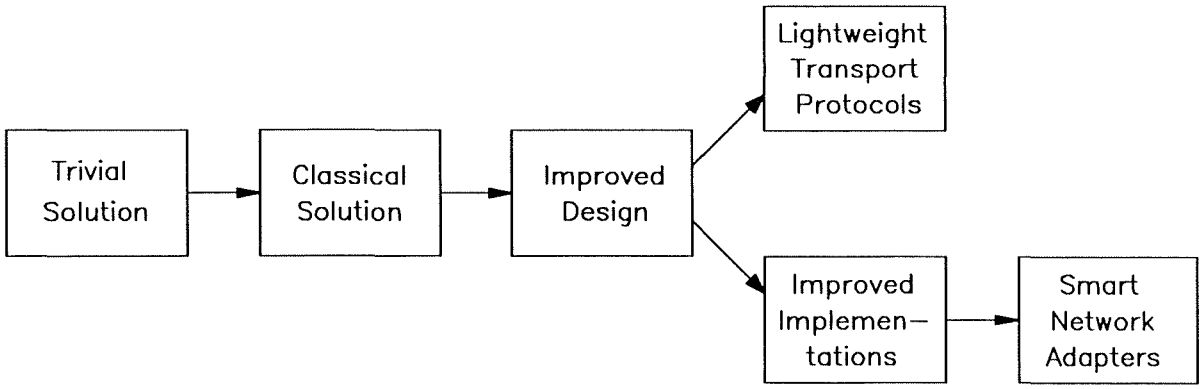
This new paradigm of computing equipments exchanging massive quantities of information with real-time constraints lead to a new requirement for the transport protocol. Beside the need for a reliable transmission of data, there is also a demand for a high throughput in this transmission.

And the problem the computer scientists have to face today is: the transport protocols, which were designed for slow networks, do not actually achieve their high throughput requirement when they control a data transmission on a fast network. Or, expressed in other words, the improvements of the links bandwidth do not naturally lead to similar improvements at the transport level.

The objective of this document.

The objective of this document is to provide a comprehensive analysis of the capacity of the transport protocols to achieve high throughput. The only class of transport protocols of interest to this discussion is the one implemented over a network layer similar to IP. This kind of network layer only provides a minimal functionality of datagrams routing, on a best-effort basis.

The first step, in the present analysis, is the presentation of this minimal network layer. It will highlight all the annoying characteristics the transport layer will have to face.



- Figure 1 -
The progression of the solutions.

A second step will introduce the description of a very simple transport protocol providing the functionality of reliable transmission of a stream of bytes. The purpose of this step is to present the basic aspects of a transport protocol.

The next discussion will deal with the classical solution used by the current transport protocols. A first attempt will be made to explain their lack of throughput by an irrational utilization of the network bandwidth.

An improved design for the transport protocols, which uses more efficiently the network layer, will then be described and analyzed. The conclusion of this analysis will be: the transport protocols can not achieve high throughput because they require too much computing resources.

A first effort on correcting this situation will then be examined. It is known as the lightweight transport protocols concept, and consists of defining brand new protocols which are supposed to naturally lead to less computing overhead. The failure of this approach will be demonstrated.

A solution allowing a real improvement of the throughput at the transport level will then be introduced. This solution consists of focussing on the implementation of the transport layer instead of its definition. Some of these implementation improvements will be discussed. It will be shown that this solution is not sufficient by itself, but indicates the right way for the subsequent investigations.

Finally, the currently best conceived solution will be presented: the smart network adapters concept. It consists of moving a part of the transport protocol handling outside the host. This allows a hardware support that helps reaching very high end-to-end throughput.

This progression inside the world of transport protocols solutions is depicted on figure 1.

The methodology of the presentation.

Some kind of a methodology will be used to present each analyzed solution:

- First, a concise presentation of the aims of the solution will delimit its scope.
- Second, the principles of this solution will be examined in depth, theoretically and/or using revealing case studies.
- Third, examples of the use of this solution will be introduced.
- Fourth, the results of the solution application will be mentioned, when they help the further discussion.
- Fifth, an analysis of this solution will be presented, to highlight its problems, failures, or merits.

Chapter I:

An analysis of the minimal network layer.

I.1. The minimal network layer.

The minimal network layer ensures the connection of a set of computers which have to communicate. These computers will be referred to by the word 'hosts'. Each host is identified in the network by its 'network address'[1].

The minimal network layer connecting all the hosts only provides a minimal functionality: it routes short messages from an originating host to a destinating host, on a best-effort basis. These short messages, which have a maximal length, are generally referred to by the word 'datagrams'.

The 'best-effort basis' is the main characteristic of the design of the minimal network layer. Its principle is that the basic functionality of datagram routing leads to a quality of service far from being perfect, but no effort of any kind is made to enhance this basic quality of service. Two important facts come from this principle:

- The basic routing leads to some casual malfunctions that will be discussed in paragraphs I.4 to I.7[2]. The network layer will not try to recover from these problems and will not warn the involved users of them, as it will even not try to detect them.
- There will be no multiplexing done by the network layer. Each network agent will only serve one user. For now, there will be no more difference made between a host, its network agent, and the unique user of this network agent.

---

1. The reader is pleased not to misread: the 'network address' is not any address of the network, but the address of a host inside a network.
2. A short description of these malfunctions may be found in [NETBLT].

So here is the specification of the only service the network layer provides, that is to say N-DATA. It is a three phases service, with a REQUEST, an INDICATION and a CONFIRMATION primitives[3]. Figure I.1 shows the usual chaining diagram of these service primitives:

- A user of the network layer calls N-DATA.REQUEST to ask its network agent to transmit some data to another user of the network layer. The main parameters of a N-DATA.REQUEST call are the network address of the destinating host, and the data to be transmitted.
- A network agent calls N-DATA.CONFIRMATION to tell a user which has issued a N-DATA.REQUEST the status of the data it wanted to be transmitted. The main parameter of a N-DATA.CONFIRMATION call is this status. It may only have one of the following meanings: 'data has been transmitted', or 'it is impossible to transmit data'.
- A network agent calls N-DATA.INDICATION to tell its user that it has received some data from another user of the network layer. The main parameters of a N-DATA.INDICATION call are the network address of the originating host, and the data that have been received.

I.2. Examples of minimal network services.

Several examples of network layers providing a service close to the minimal one exist. The most popular one is undisputably the Internet Protocol (IP)[4], in the Department of Defense (DOD) world. Nowadays, almost all computers are sold with an implementation of IP. The Open Systems Interconnection (OSI) model also defines such a network service: the Connection-Less Network Service (CLNS)[5].

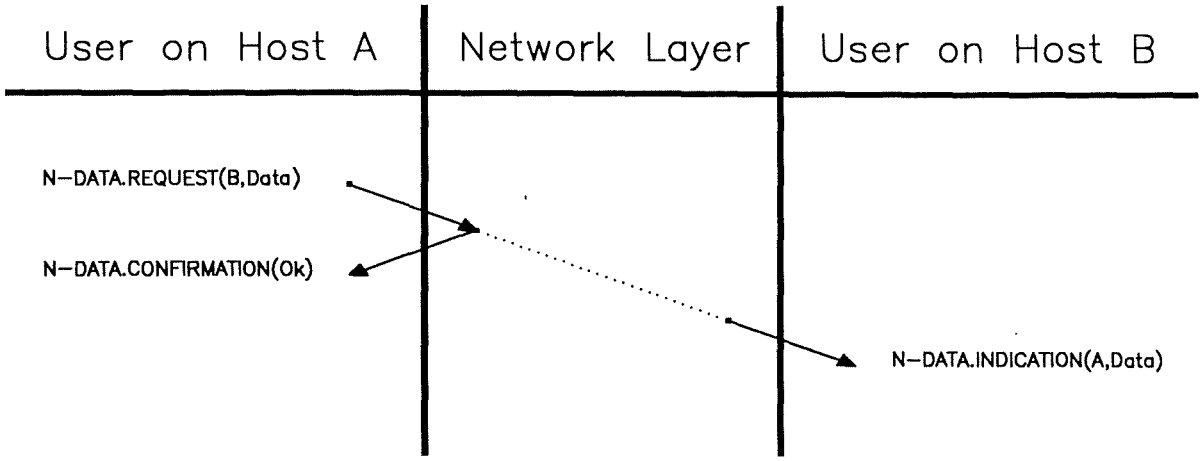
I.3. Implementation of the minimal network layer.

The more general implementation of the minimal network service described in paragraph I.1 can be provided by a mesh network of store-and-forward routers. These routers exchange datagrams, which are mainly composed of:

- the network address of the host which originated the datagram,

---

3. Here, only OSI notations and OSI terminology are used. But this is not the expression of the OSI solution of the minimal network layer.
4. A complete description of IP can be found in [RFC-791].
5. A complete description of CLNS can be found in [ISO-8473].



- Figure I.1 -
The usual chaining diagram of the N-DATA service.

- the network address of the host the datagram is destinated to,
- the data to be delivered.

The set of routers is organized as a mesh network. This means that each router is directly connected to an arbitrary set of other routers, without any more topological constraint than the basic one: connectivity. The connectivity constraint ensures that at least one route can be found between any two hosts. With the same idea of generality, there is no constraint defined for the characteristics of the links connecting neighbouring routers.

This name of 'store-and-forward router' comes from the way they are designed. Each router manages a buffer space, using the First-In-First-Out (FIFO) model. Two processes have an access to this data structure:

- The first process treats all the datagrams coming from the links the router is connected to. It just stores them in the buffer.
- The second process treats each datagram stored in the buffer. It extracts the destination address of a datagram from its header, and from this address it computes which router it has to forward the datagram to. When a router is selected, the process transmits the datagram on the appropriate link, as soon as this last is ready to send.

The network agent of each host is also a router, but with a slightly different routing algorithm than the one used in the normal routers. It must also detect the datagrams which are

destinated to its local user. These datagrams are not to be forwarded to another router, but to be delivered to the local user.

To offer a good routing service, the routers may exchange informations about the components of the network. The routing algorithm may take into account these parameters in its routing decision. So routing has a dynamic dimension: two consecutive datagrams, originated by a single host and destinated to another single one, may use different routes.

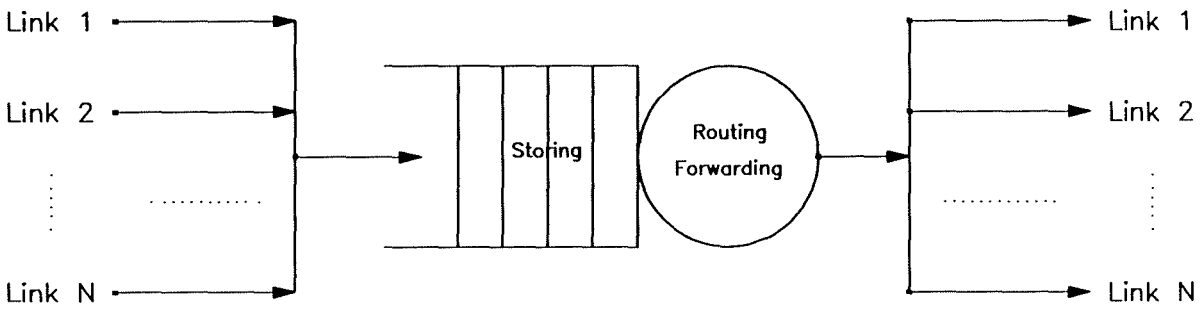
For the purpose of analysing the inherent malfunctions in the minimal network service, it is interesting to modelize a router as a stochastic waiting-queue, as shown in the figure I.2. The clients will be the datagrams which come from the links numbered 1 to N. On arrival, these clients enter a FIFO waiting-queue where they wait until they are serviced. This service is the routing decision and the forwarding to one of the outgoing links.

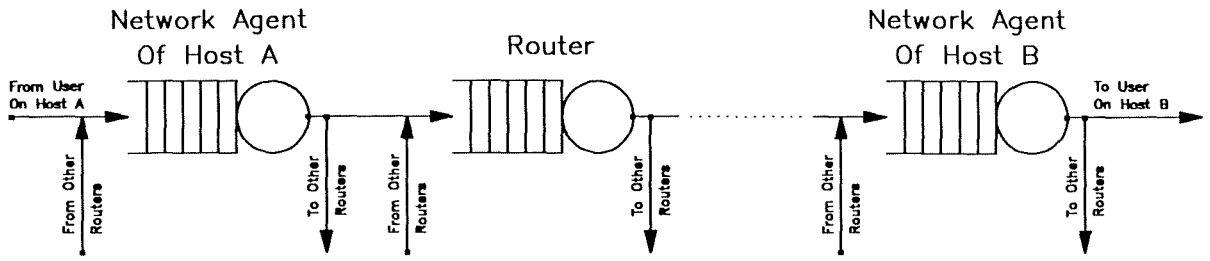
In the same way, the route a datagram uses for going from an originating host to a destinating one can be modelized as a network of waiting-queues, as shown in figure I.3. A datagram travels from router to router, which can be modelized as waiting-queues, as it has just been shown.

I.4. Datagrams are sometimes lost.

In the preceding paragraph, a router has been modelized as a stochastic waiting-queue. The two main parameters of such a queue are its client's interarrival time distribution, and its service time distribution. From these two distributions, the stochastic theory allows to compute the distribution of another

- Figure I.2 -
The stochastic model of a router.





- Figure I.3 -
The stochastic model of a route.

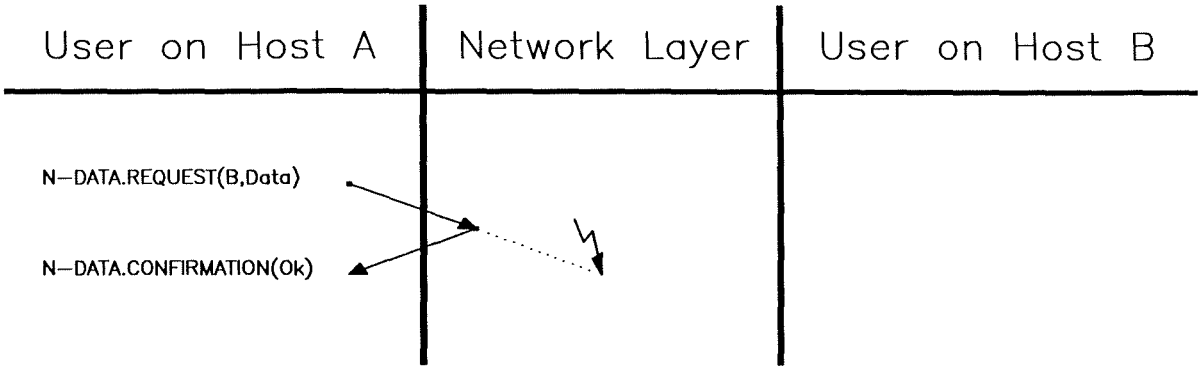
random variable: the number of clients present in the waiting-queue. From this distribution, it is fair to compute the probability of having more than a given number of clients present in the waiting-queue. The fact is: for any x, the probability of having more than x clients present in the waiting-queue is non-null.

To illustrate this result, the following possible scenario can be imagined. A router is connected to a high-bandwidth link, as well as to a low-bandwidth one. For some period of time, datagrams are coming, using all the capacity of the fast link, and all of them are to be forwarded to the slow link. The longer this period of time is, the higher the number of stored datagrams is.

The problem is that a router only handles a finite amount of buffer space, and thus can only store a finite number of datagrams. But it has just been shown the possibility that, at some moment, the router needs to store more than this amount of datagrams. The question is then: what must do the router when it receives a datagram and there is no available buffer space to store it? According to the best-effort basis concept, the router can solve this problem by straightforwardly discarding the datagram[6].

The consequence of this problem is described by the figure I.4. It represents another possible chaining diagram for the N-DATA service. A user issued a N-DATA.REQUEST to its network agent. It was responded a N-DATA.CONFIRMATION. But the expected receiving user never received the associated N-DATA.INDICATION. This is the problem of lost datagrams.

6. Another solution is to drop one of the buffered datagrams, and add the incoming one to the end of the queue. But, any way, a datagram is lost.



- Figure I.4 -
The network layer sometimes drops datagrams.

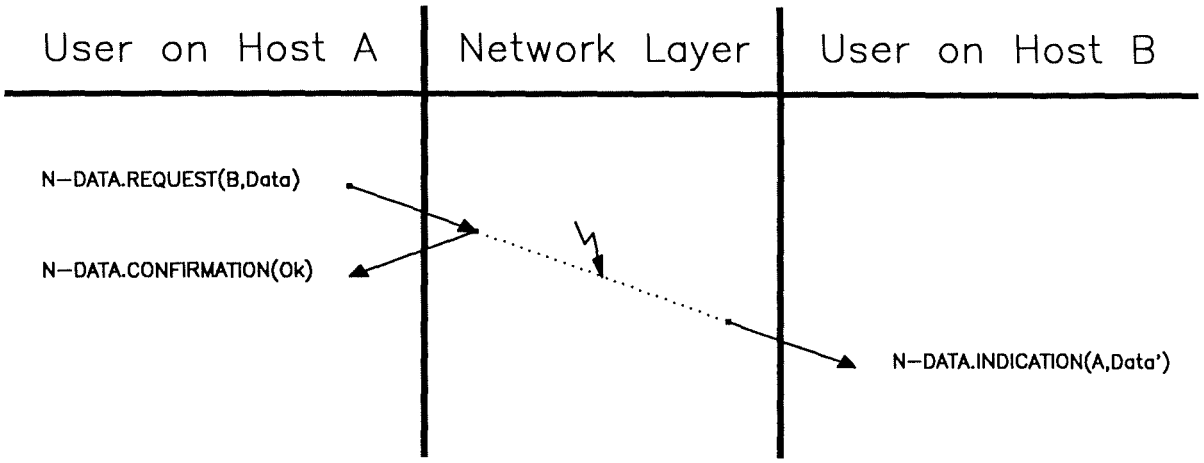
I.5. Datagrams are sometimes corrupted.

As stated in the paragraph I.2, there is no constraint for the links connecting the routers. Some of them may implement an error recovery mechanism, and some of them may not. However, a link will always be characterized by its error-rate. This rate will be the residual error-rate of the recovery mechanism if the link implements such a mechanism, and the basic transmission error-rate of the physical layer in the opposite case.

When a datagram comes to a router, it is received by some device. The router has then to move the datagram from the receiving device to its main memory. When the routing decision has been taken, the datagram is moved from the main memory of the router to one of its sending devices. All these copy operations can also lead to errors, and so the routers will also be characterized by an error-rate.

From all these error-rates, it is possible to compute an error-rate for the network layer. And it is likely that this rate will be non negligible.

According to the best-effort principle, the network layer will not deal with this problem. The consequences of that fact is shown on figure I.5. It also represents another possible chaining diagram for the N-DATA service. A user issued a N-DATA.REQUEST to its network agent with a given value for the data, and this user was responded a N-DATA.CONFIRMATION. But the receiving user received the corresponding N-DATA.INDICATION with another value for the data. This is the problem of corrupted datagrams.



- Figure I.5 -
The network layer sometimes corrupts datagrams.

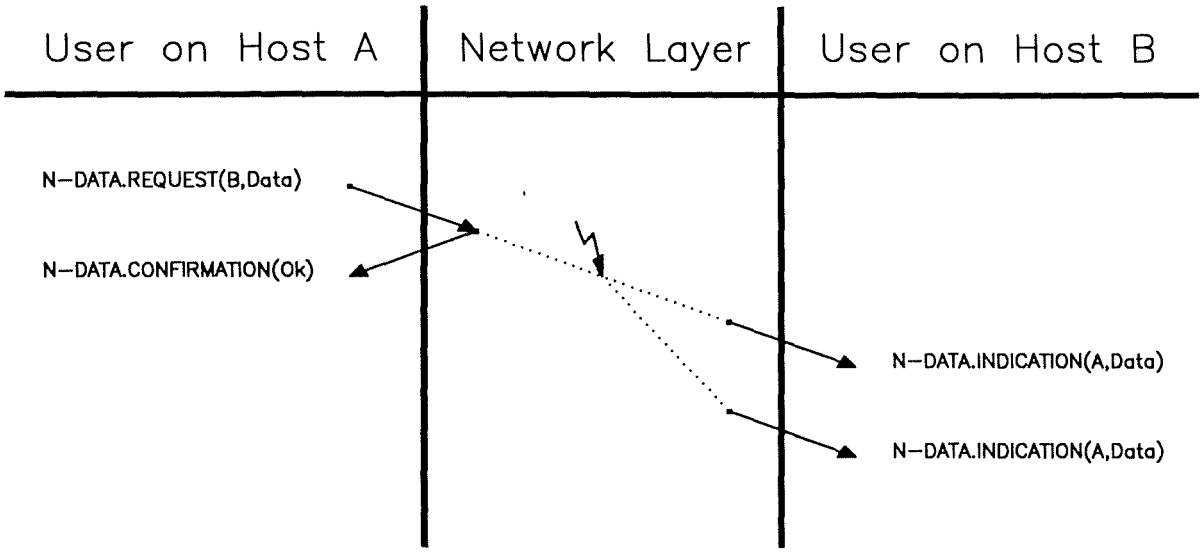
I.6. Datagrams are sometimes duplicated.

As stated in the preceding paragraph, some of the links connecting two routers may implement some error-recovery scheme. Such a link ensures its users that any message it is given is correctly delivered to its destinator.

The problem is that the design of these error-recovery schemes[7] sometimes leads the destinator to get two or more copies of the same message. And so, in the network, when a router forwards a datagram to another router, it is not impossible that the latter receives this datagram more than once.

According to the best-effort principle, the network layer will not deal with this problem. The consequences of that fact is shown on figure I.6, where another possible chaining diagram for the N-DATA service is represented. A user issued a N-DATA.REQUEST to its network agent, and was responded a N-DATA.CONFIRMATION. But the destinating user received more than once the corresponding N-DATA.INDICATION. This is the problem of duplicated datagrams.

7. In fact, it is often a link level retransmission on timeout scheme, which is also used at the transport level. Such a scheme will be discussed in the next chapter.



- Figure I.6 -
The network layer sometimes duplicates datagrams.

I.7. Datagrams are sometimes re-ordered.

Another malfunction of the minimal network layer can be highlighted by analysing its latency characteristics. The latency of the network is the elapsed time between the moment a user issues a N-DATA.REQUEST and the moment the receiving user gets the corresponding N-DATA.INDICATION.

In paragraph I.3, a route has been modelized as a network of stochastic waiting-queues. From the clients' interarrival time distributions, and the service time distributions of all these stochastic waiting-queues, the stochastic theory allows to compute the distribution of another random variable: the time a client spends in the stochastic network.

Also in this paragraph I.3, it has been stated that the routing may have a dynamic dimension: the route a datagram uses to get from one user of the network layer to another one may change from time to time. And so the network of stochastic waiting-queues modelizing the route between two users of the network layer may also change from time to time. The consequence is that the distribution of the random variable describing the time a client spends in the stochastic network varies along time.

The transposition in the minimal network layer of the 'time a client spends in the stochastic network' is nothing else than the 'network latency'. The network latency is thus modelized by

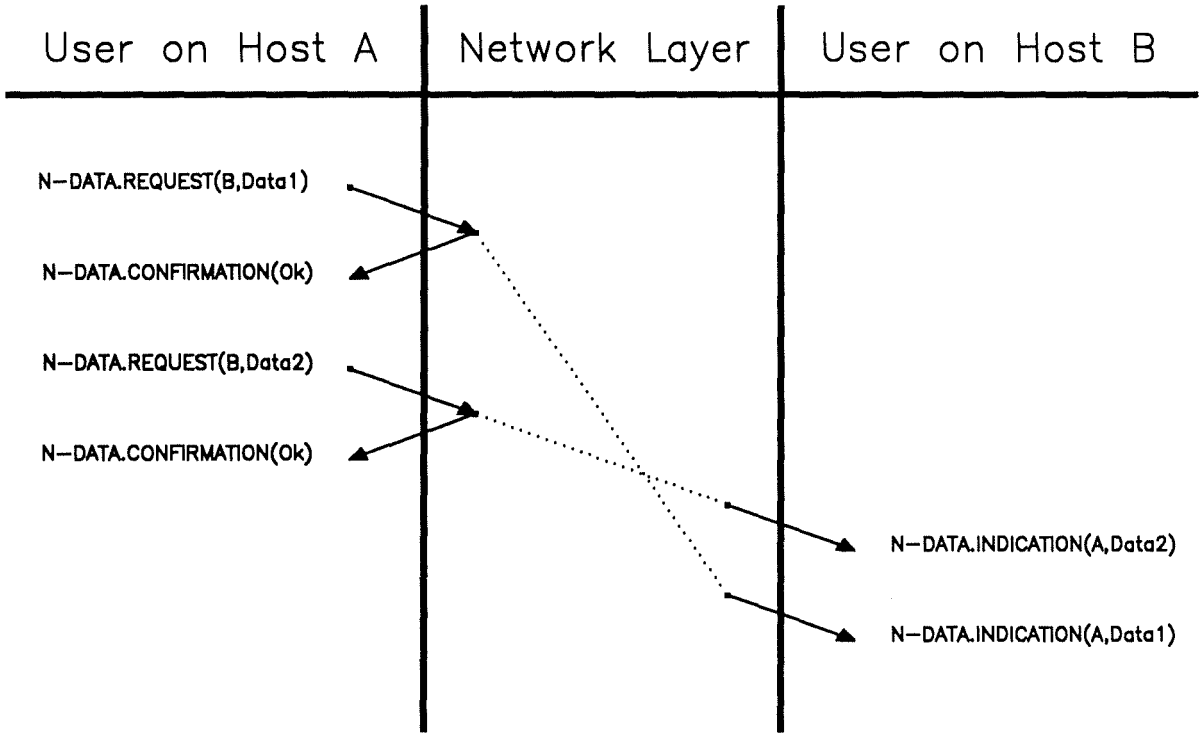
a random variable. The conclusion is then: the network latency varies along time.

This fact has a very annoying consequence, as shown in figure I.7: a user issued two consecutive N-DATA.REQUESTs to its network agent, and was responded two N-DATA.CONFIRMATIONs. But the receiving user has got the N-DATA.INDICATION corresponding to the second N-DATA.REQUEST before it has got the one corresponding to the first N-DATA.REQUEST. This is the problem of datagrams re-ordering.

I.8. Conclusion.

The minimal network layer only provides a single service to its users: the delivery, on a best-effort basis, of a datagram to another user. This best-effort basis leads to some problems which are: datagrams re-ordering, duplication, corruption, and loss.

- Figure I.7 -
The network layer sometimes re-orders datagrams.



So, for designing a transport layer with the functionality of point-to-point reliable transfer of a stream of data, there is a need to find some mechanisms which will detect these problems and which will recover from them.

Chapter II:

A Trivial Solution.

II.1. Aims of this solution.

The aim of the trivial solution[1] is to build a simple transport protocol providing the functionality of 'reliable end-to-end user-data transmission', in spite of the problems of the minimal network layer that have been discussed in chapter I.

II.2. Principles of this solution.[2]

* Dealing with multiplexing.

There will be a need to allow more than a single user of the transport layer in each host. The problem is that the network layer only allows a unique user in each host[3].

The unique user of the network layer will in fact be the transport agent of the host. The latter will serve multiple users, that will be referred to by the word 'entities'. There will be a need to distinguish all the entities a transport agent serves. The easiest way to do this is to allocate to each entity an identifier local to the transport agent. This identifier is often referred to by the word 'port', and has often a numeric value.

The service of the transport layer will thus be, for example, to transfer user-data from the entity on port 'ps' of host 'hs' to the entity on port 'pd' of host 'hd'. In this example:

- 'hs' is the address of the host running the sending entity,

---

1. The author does not want to use the word 'trivial' in its pejorative sense. He chose this word because of the name of the protocol implementing the trivial solution: the Trivial File Transfer Protocol, or TFTP. Instead, in this context, 'trivial' means 'basic', or 'straightforward'.
2. A description of the trivial solution can be read in [NETBLT].
3. This has been discussed in paragraph I.1.

- 'ps' is the port number of the sending entity,
- 'hd' is the address of the host running the destinating entity,
- 'pd' is the port number of the destinating entity.

To implement the service provided by the transport layer, the transport agents will have to exchange informations using datagrams. These datagrams are referred to by the expression 'Transport Protocol Data Units', or TPDUs.

At one moment, a transport agent will be involved in several user-data transfers. The problem is that when it sends a TPDU to another transport agent, or when it receives a TPDU from another one, the interaction with the network layer only involves this TPDU and the address of the peer transport agent. But this is not sufficient to identifies which transfer this TPDU concerns. So, each TPDU will include at least two fields:

- the Source port field, which value is the port allocated to the sending entity,
- the Destination port field, which value is the port allocated the destinating entity.

*** Dealing with the maximal length of the datagrams.**

A very interesting functionality to provide to the entities is the transfer of data without any constraint on the data length. But the problem is that the transport agents will have to exchange TPDUs using datagrams, which have a fixed maximal length[4].

The solution is the transport agent to split the user-data in several parts that fit the maximal datagram length. This is called the packetization of the user-data. So the transport agent of the sending entity sends a sequence of TPDUs, each containing a part of the user-data. These TPDUs will be referred to by the expression 'DATA-TPDUs'. They will be composed of the four following fields:

- the Source port field, which value is the port allocated to the sending entity,
- the Destination port field, which value is the port allocated to the destinating entity,
- the Data field, which value is a part of the transferred user-data,
- the Flags field[5], which each bit marks a particular boolean condition, as 'end-of-message'[6].

The transport agent of the destinating entity rebuilds the user-data from the parts it receives in the DATA-TPDU before delivering it to this destinating entity.

---

4. This has been discussed in paragraph I.1.
5. For clarity reasons, this field will never be shown on the figures.
6. This bit would only be set in the last DATA-TPDU of a user message.

*** Dealing with datagrams re-ordering.**

An important problem of the minimal network layer is that it does not keep the sequence of datagrams[7]. But, to rebuild the user-data from the parts it receives, the transport agent of the receiving entity needs to get the DATA-TPDUs in the same order the transport agent of the sending entity delivered them to the network layer.

A very simple solution is this one: the transport agent of the sending entity only sends a DATA-TPDU containing a part of the user-data when it is sure that the transport agent of the destinating entity actually got the DATA-TPDU containing the preceding part of their user-data.

But this solution leads to another problem: as the transport agent of the sending entity and the one of the destinating entity are not located in the same host, they do not share the same memory. So a transport agent can only get an information about another one if this last sends a TPDU containing the appropriate information to the former.

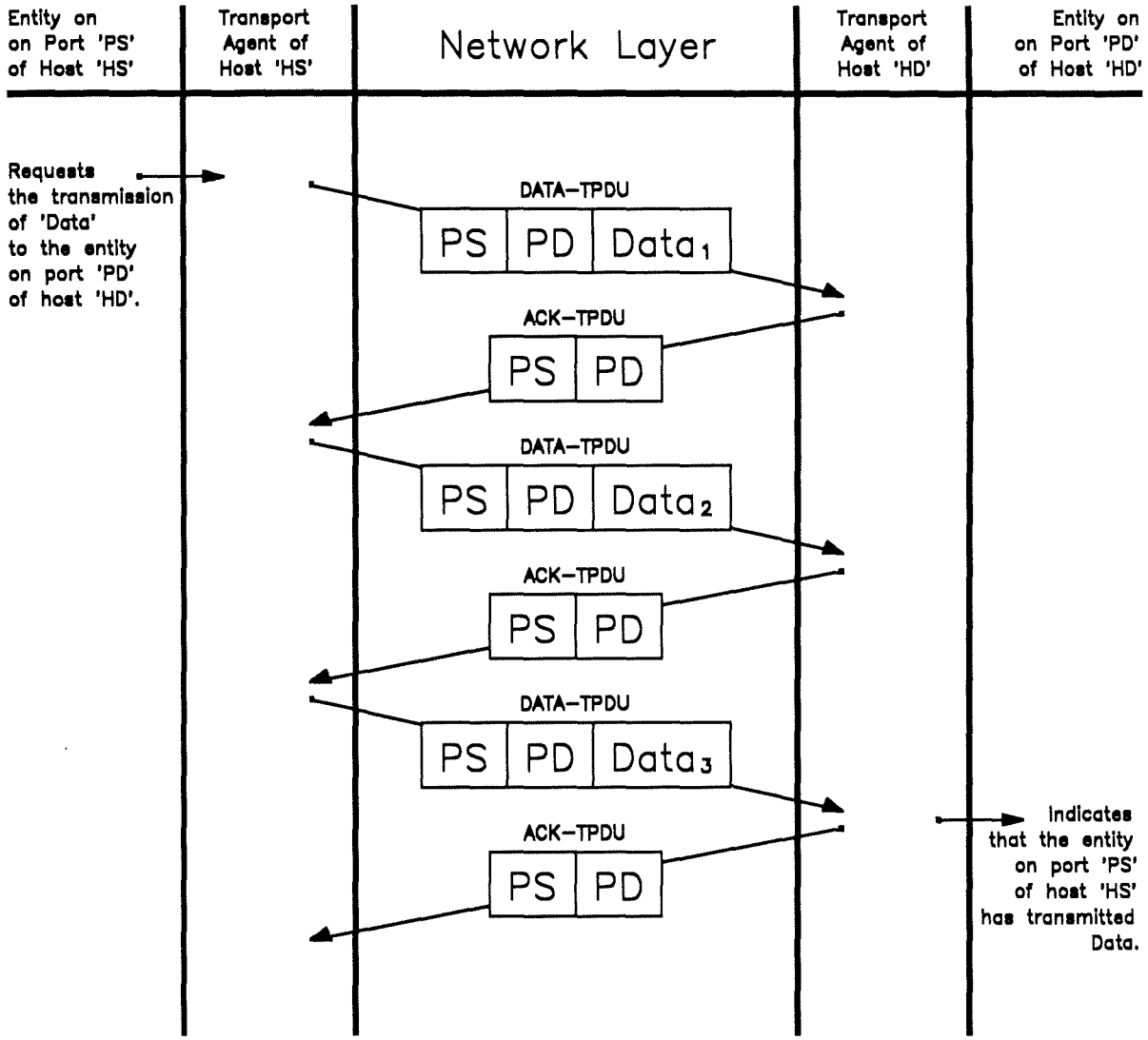
Here is a scheme to make the described solution work. When the transport agent of the destinating entity receives a DATA-TPDU, it sends a TPDU back to the transport agent of the sending entity, to allow the latter to transmit the next DATA-TPDU. This scheme is called the 'acknowledgement principle'. The TPDU the transport agent of the destinating entity sends back to the transport agent of sending one is referred to by the expression 'ACK-TPDU'. An ACK-TPDU will be composed of the two following fields:

- the Source port field, which value is the port allocated to the sending entity,
- the Destination port field, which value is the port allocated to the destinating entity.

The usual transfer of user-data by the transport layer is shown on figure II.1. The sending entity delivers some data to its transport agent and wants this data to be delivered to another entity. This data has to be packetized in three parts, that is to say Data1, Data2 and Data3. The transport agent of the sending entity transmits a first DATA-TPDU containing Data1 to the transport agent of the destinating entity. The latter sends back a ACK-TPDU to the transport agent of the sending entity. When the transport agent of the sending entity has received this ACK-TPDU, the scenario described for Data1 is repeated for Data2 and then for Data3. When the transport agent of the destinating entity has received the DATA-TPDU containing Data3, it is able to re-assemble the data the sending entity delivered to its transport agent, and so is able to deliver this data to the destinating entity.

---

7. This has been discussed in paragraph I.7.



- Figure II.1 -
Dealing with datagrams re-ordering.

*** Dealing with datagram loss.**

The minimal network layer sometimes drops a datagram[8], without warning the sender or the expected receiver of that fact. So when a transport agent sends a TPDU to another one, it only knows that this TPDU was transmitted, but it does not know if it was actually received or not. On the other hand, when a transport agent is the expected receiver of a TPDU and when this TPDU is lost by the network layer, it does not even know that another transport agent tried to send it a TPDU.

8. This has been discussed in the paragraph I.4.

The only available information for a transport agent about this problem is the following one. When it receives a TPDU concerning the transfer of some user-data between two entities, it knows that the last TPDU it issued for that transfer was correctly received. All other information is only based on speculation.

But there is an interesting speculation a transport agent can make: when it sends to its peer a TPDU concerning a transfer of user-data, it will receive a TPDU from the latter in a near future. When it sends a ACK-TPDU, the transport agent of a destinating entity cannot really speculate about the delay before it will get the next DATA-TPDU. As a matter of fact, the transport agent of the sending entity will only send this DATA-TPDU when it has got the ACK-TPDU and it has data to send. But no transport agent can speculate about the delay before this last condition is fulfilled: it depends on the behaviour of the sending entity. On the other hand, the transport agent of the sending entity knows that its peer has no reason to delay the sending of a ACK-TPDU: it will transmit this TPDU as soon as it receives the corresponding DATA-TPDU.

The detection of lost datagrams will thus be left to the transport agent of the sending entity. If, some reasonable time after it has delivered a DATA-TPDU to the network layer, it does not receive the corresponding ACK-TPDU, it can speculate that this DATA-TPDU or this ACK-TPDU has been lost by the network layer. And in this case, it has nothing else to do than retransmitting the lost DATA-TPDU. This scheme is referred to by the expression 'retransmission on timeout'.

The most important question is now: what is a reasonable value for the timeout delay? For that matter, the story of a DATA-TPDU and its corresponding ACK-TPDU must be analysed:

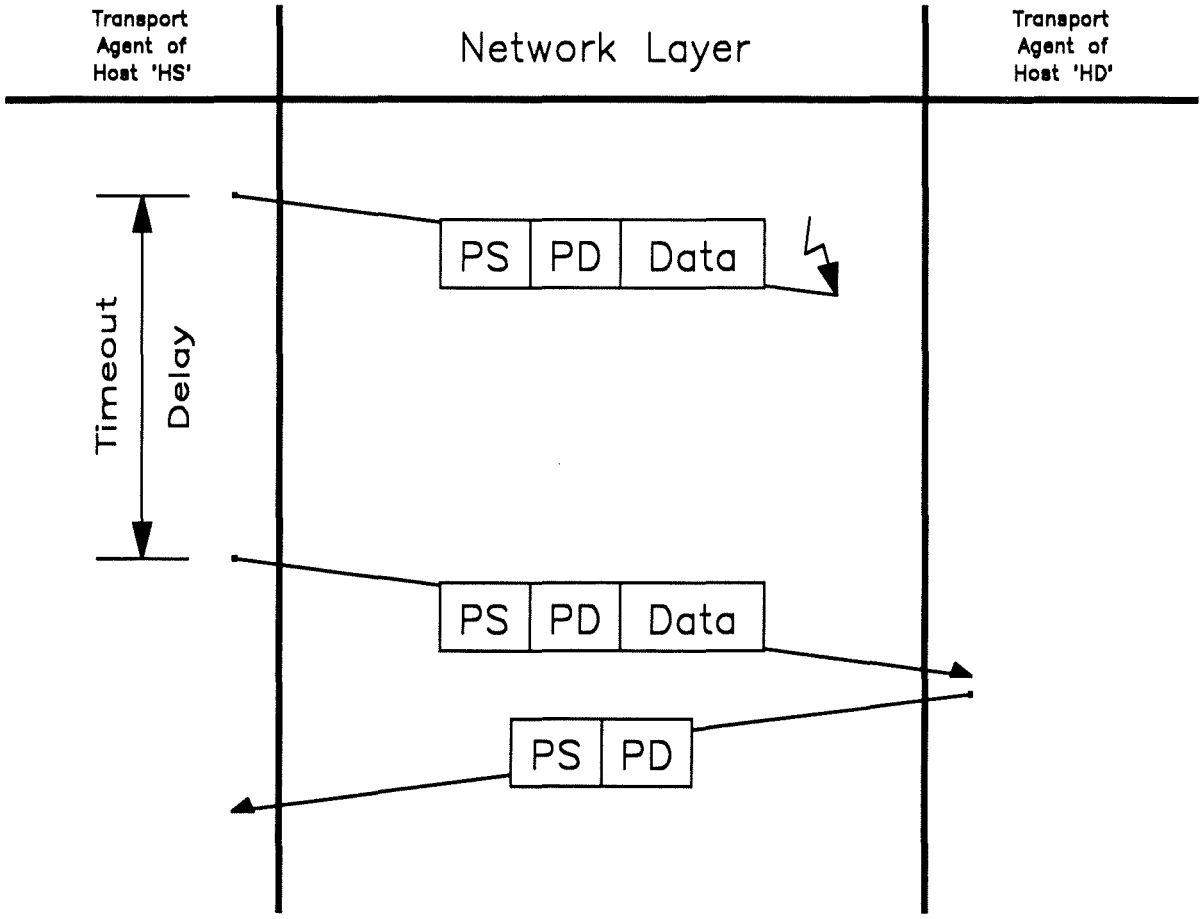
- The DATA-TPDU is routed from the transport agent of the sending entity to the transport agent of the destinating entity. This costs a network latency.
- The DATA-TPDU is received by the transport agent of the destinating entity, an ACK-TPDU is built and delivered to the network layer. This costs a negligible amount of time.
- The ACK-TPDU is routed from the transport agent of the destinating entity to the transport agent of the sending entity. This costs a network latency.
- The ACK-TPDU is received by the transport agent of the sending entity. This also costs only a negligible amount of time.

So the reasonable timeout delay will be the result of the sum of these four components. This sum is often referred to by the expression 'Round Trip Delay' or RTD. This name comes from the fact that it is nearly equal to the time a datagram would spend to go from a sending host to a receiving one and then another datagram to go back from the receiver to the sender.

The main problem is that the RTD is mainly composed of network delays, which have variable and unpredictable values[9]. The transport agent of the sending entity will have to use an estimation of this RTD as the timeout delay. A fair way to compute an estimation of the RTD is to take into account the measured values of RTD for a few past TPDU exchanges, and to assume that the RTD will only vary smoothly.

An example of exchange between transport agents illustrating the retransmission on timeout is shown on figure II.2. The transport agent of the sending entity has just received the ACK-TPDU corresponding to the latest DATA-TPDU it has transmitted. So it sends the DATA-TPDU containing the next part of the user data. But this TPDU is lost by the network layer. When the timeout delay expires, the transport agent retransmits the lost DATA-TPDU.

- Figure II.2 -
The retransmission on timeout principle.



9. This has been discussed in paragraph I.7.

The annoying consequence of using a 'retransmission on timeout scheme' is that the transport agent of the sending entity sometimes transmits twice the same DATA-TPDU, even if there is no reason to do it. This happens in two cases:

- when the estimation of RTD leads the timeout delay to be shorter than the actual RTD,
- when a ACK-TPDU is lost.

*** Dealing with duplicated datagrams.**

When a transport agent delivers a TPDU to the network layer, it is not impossible that the destinating transport agent gets more than once the copy of this TPDU[10]. Furthermore, as it has just been shown, the mechanism to recover from lost TPDUs sometimes leads the transport agent of the destinating entity to get several copies of the same DATA-TPDU.

A simple solution to cope with this problem is to add a field to each exchanged TPDU, which value identifies the TPDU. A transport agent would then only treat the TPDUs with the expected identifier, and discard the other ones.

A good identifier for a DATA-TPDU is the sequence number of the data it contains, relative to the user-data that is being transferred. A good identifier for a ACK-TPDU is the same identifier as the corresponding DATA-TPDU.

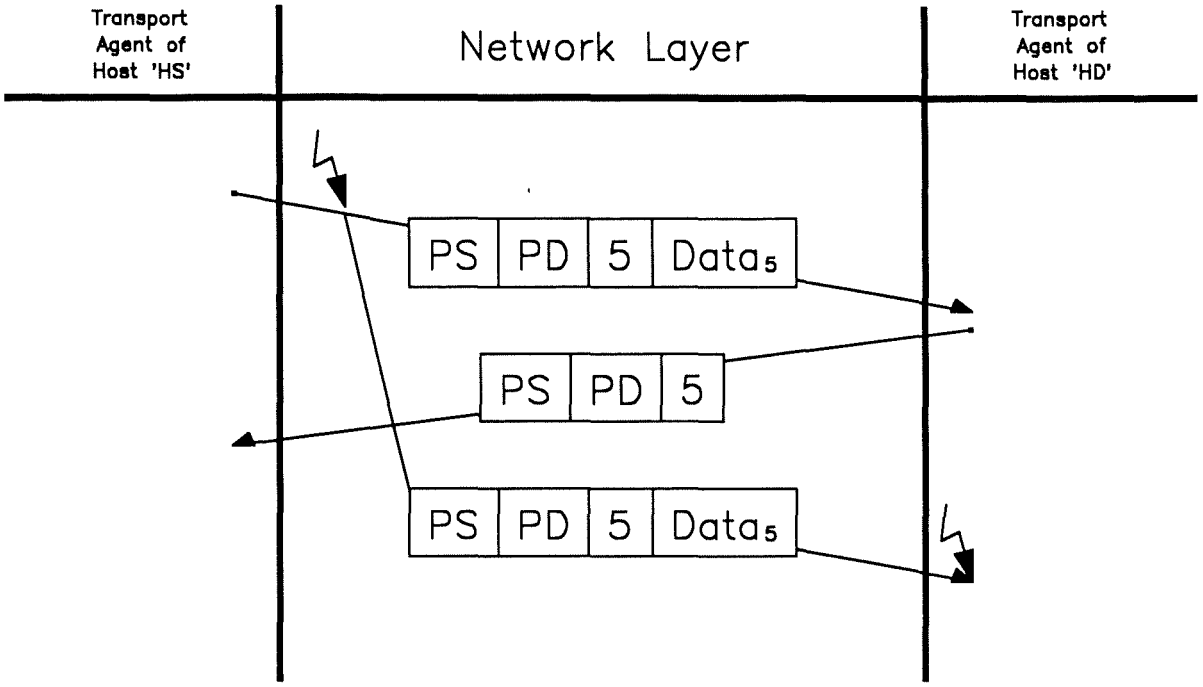
Figure II.3 shows an exchange between two transport agents. The transport agent of the sending entity has just received a ACK-TPDU corresponding to the fourth part of the user-data. So it transmits a DATA-TPDU containing the fifth part of the user-data. The transport agent of the destinating entity gets a first copy of this DATA-TPDU and sends back the corresponding ACK-TPDU. Then this transport agent gets another copy of the DATA-TPDU containing the fifth part of the user-data. But as it is expecting the sixth part, it discards this DATA-TPDU as a duplicate.

*** Dealing with datagram corruption.**

When a transport agent delivers a TPDU to the network layer, it is not impossible that the destinating transport agent gets a corrupted copy of this TPDU[11]. In other words: when a transport agent gets a TPDU, it is not really sure that the value of each field of this TPDU is actually the one the sending transport agent expected to be received.

---

10. This has been discussed in paragraph I.6.
11. This has been discussed in paragraph I.5.



- Figure II.3 -
Dealing with duplicated TPDUs.

There will be only scarce problems if a control field is corrupted, as there will be a very little chance that the transport agent accepts such a corrupted TPDU. This transport agent can find:

- that the mentioned destinating entity is not involved in any transfer at the moment,
- that the mentioned destinating entity is currently involved in a transfer, but not with the mentioned sending entity,
- that the identifier mentioned in the TPDU is not the expected one for the transfer between the mentioned sending and receiving entities.

But if such a corrupted TPDU is accepted and treated as a correct one, the consequence is very annoying: the synchronization may be lost between the transport agents involved with this transfer of user-data. This can lead to discard correct TPDUs, to send TPDUs at a wrong time, or not to retransmit a lost TPDU.

The problem of corrupted TPDUs is more clear when only the Data field of a DATA-TPDU is corrupted. There is no chance to discover this corruption with the existing mechanisms. So all these DATA-TPDUs with a corrupted Data field are accepted and treated. When this happens, the user-data which is re-assembled

and delivered to the destinating entity is not the one the sending entity delivered to its transport agent. But the transport layer has to offer a reliable user-data transfer!

The solution to this problem is to add another field to each TPDU. The value of this field would be redundant with the content of the other fields of the TPDU. The receiving transport agent would then re-compute this redundant information from the value of the fields in the received TPDU. It would then check this re-computed redundant information against the one found in the received TPDU. If the specification of the redundant information is well chosen, there is a very little chance that there is a match when the received TPDU is corrupted.

When a transport agent receives a corrupted TPDU, it may straightforwardly drop it. The lost TPDU recovery mechanism ensures that a correct version of this TPDU will be soon retransmitted.

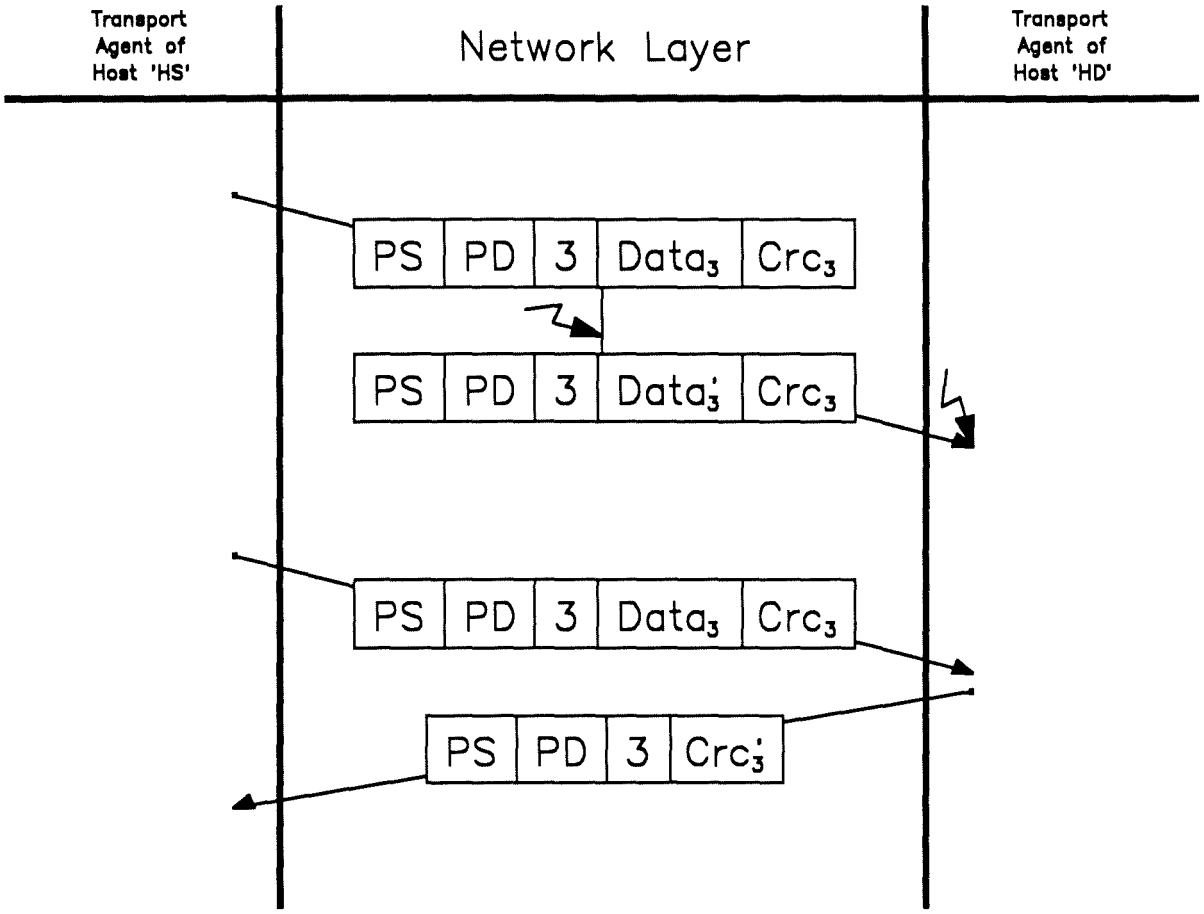
A very popular redundant information is the sum of all the words the TPDU is composed of. This kind of redundant information is often referred to by the word 'checksum'. It gives rather good results, it is very easy to implement, and all other redundant informations which are function of the entire TPDU content need at least as much computing resource. For now, the redundant information will always be referred to using the word 'checksum'[12].

Figure II.4 illustrates the checksum mechanism. The transport agent of the sending entity has just received the expected ACK-TPDU. So it transmits the next DATA-TPDU. But the Data field of this TPDU is corrupted by the network layer. When the transport agent of the destinating entity receives this TPDU, the verification of the checksum fails, and the TPDU is discarded. When the timeout delay expires, the transport agent of the sending entity retransmits the DATA-TPDU.

II.3. Example of the trivial solution.

The Trivial File Transfer Protocol (TFTP)[13], is not really a transport protocol, but an application layer protocol. This protocol is specified to rely on the transport protocol called User Datagram Protocol (UDP)[14]. UDP itself relies on IP, and only deals with multiplexing: its only service is similar to the minimal service of the network layer; but this service is

12. In fact, the checksum is a so popular redundant information that all redundant informations used to check the integrity of data is also referred to as a 'checksum', even if its mechanism does not involve any summation.
13. A complete description of TFTP can be found in [RFC-783].
14. A complete description of UDP can be found in [RFC-768].



- Figure II.4 -
Dealing with corrupted TPDUs.

provided between transport entities and not between hosts. So, in fact, TFTP has to deal with exactly the same problems as a transport protocol which relies on IP, except for the multiplexing point of view. And, just as it can be half seen from its name, it uses the trivial solution that as just been described.

Diskless workstations and X-terminals often use TFTP to load the kernel of their operating system, at boot time: the driver of TFTP, as well as the driver of IP are simple enough to be firmware coded.

II.4. Analysis of the trivial solution.

The aim of the trivial solution was to provide a given service, without any further consideration[15]. So in this chapter, there has never been any attention paid to throughput considerations. And, as it can be easily imagined, the main problem of the trivial solution is its lack of throughput.

To calculate the maximal theoretical throughput of the trivial solution, a particular case will be considered:

- The two communicating entities are the only ones that exchange data. So all the available bandwidth of the network is dedicated to this user-data transfer.
- The topology of the network allows to find a route for this communication, which can afford a maximal bandwidth of BW before a datagram is lost, as a result to a router congestion.
- The maximal allowed length for the datagrams makes the DATA-TPDUs all have a length equal to DL.

The sending transport agent sends a DATA-TPDU and then waits a RTD time to get an ACK-TPDU. To send the DATA-TPDU, it takes an amount of time equal to:

$$\text{Time to send a DATA-TPDU} = \frac{DL}{BW}.$$

So the sending transport agent transmits an amount of data equal to DL, during a time equal to (DL/BW)+RTD. So the throughput of the trivial solution is:

$$\text{Throughput} = \frac{DL}{DL \;/\; BW + RTD}.$$

By using some elementary calculus, this formula is found to be equivalent to the following one:

$$\text{Throughput} = \frac{DL}{DL + RTD * BW} * BW.$$

15. This has been discussed in paragraph III.1.

This last formula shows that, with the trivial solution, only a part of the available bandwidth is used. And the longer the RTD is, or the faster the network works, the shorter this part is.

An extremely bad case would be a satellite network, where:
- The maximal length of a datagram (DL) would be a few kilobytes.
- The available bandwidth (BW) would be in the order of a hundred megabits per second.
- the RTD would be in the range of the second.

In this case, the trivial solution would only use about one ten thousandth of the available bandwidth.

On the same idea, the trivial solution only uses the computing resource to receive ACK-TPDU, to react to timeouts, and to prepare DATA-TPDUs. And most of the time, it merely waits, i.e. it does not use CPU time. So the trivial solution only uses a very short part of the available CPU time.

For short, the two main resources for communicating:
- the network bandwidth for transmitting informations,
- the computing resource for driving the protocol,

are wasted when using the trivial solution.

II.5. Conclusion.

It is possible to find simple mechanisms to correct the malfunctions of the minimal network layer, in a way to provide a transport service allowing a 'reliable end-to-end user-data transmission'. But these simple mechanisms do not allow to have a data transfer with a good throughput, mainly because a lot of resources are wasted when the transport agent of the sending entity waits the acknowledgement for a DATA-TPDU it has just transmitted.

Nevertheless, it is interesting to carefully study this solution, as it introduces almost all basic mechanisms used by a transport layer relying on a minimal network layer.

But there is a need to find better solutions, allowing a better throughput. And these solutions will have to focus on avoiding the sending transport agent to spend some time at doing nothing else than waiting for an event.

Chapter III:

The classical solution.

III.1. Aims of this solution.

The aim of the classical solution[1] is to be a quite simple solution for a transport layer to lead to a better throughput than the trivial solution. It will meet its throughput requirement by allowing the transport agent of the sending entity to continuously transmit DATA-TPDUs. And it will meet its simplicity prerequisite by having its design very close to the one of the trivial solution.

III.2. Principles of the solution.[2]

*** General principle.**

In the trivial solution, the reason why the transport agent of the sending entity spends time waiting for an ACK-TPDU is the need to deal with the network problem of datagrams re-ordering[3]. In this trivial solution again, the network problem of datagrams duplication is solved by adding a sequence number to each TPDU[4].

An interesting observation is the fact that this sequence number can also solve the network problem of datagrams re-ordering. When the network layer delivers a DATA-TPDU to the transport agent of the receiving entity, the latter knows from the sequence number of this DATA-TPDU which part of the initial message it contains. So this initial message can be rebuilt, even if the DATA-TPDUs do not come to the transport agent of the receiving entity in the appropriate order. The conclusion is

---

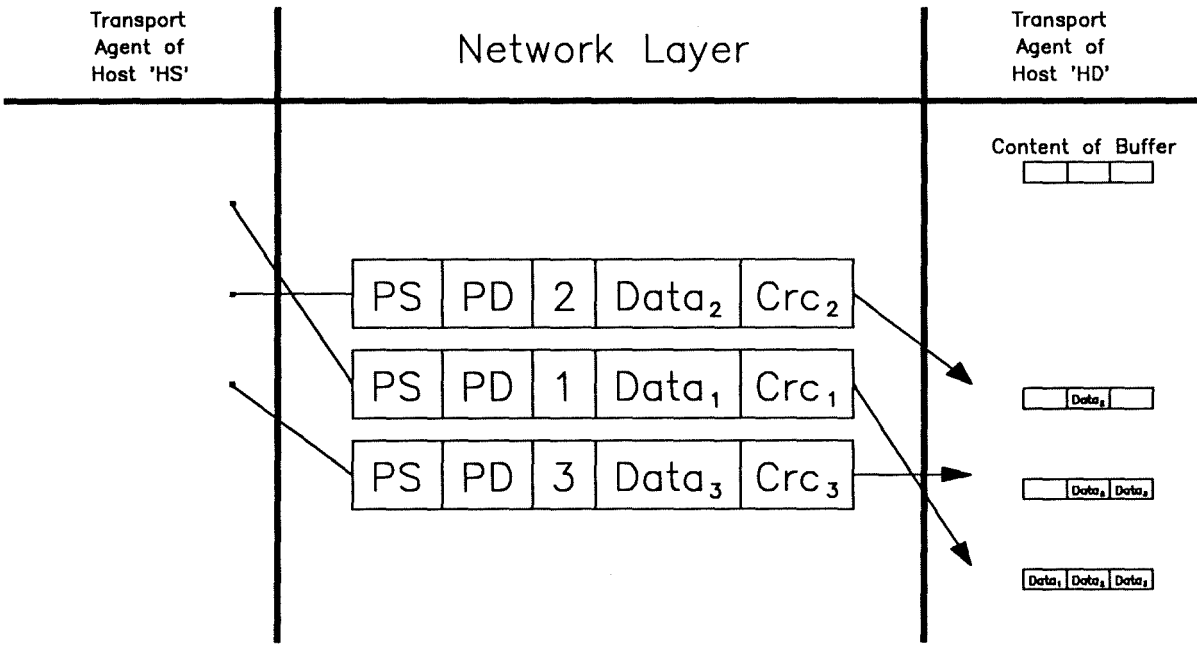
1. The author uses the expression 'classical solution' in reference to the fact that this solution is currently the most spread one, with the transport protocols TCP and TP-4.
2. A description of the classical solution can be read in [NETBLT].
3. This has been discussed in paragraph II.2, under the title "Dealing with datagrams re-ordering".
4. This has been discussed in paragraph II.2, under the title "Dealing with duplicated datagrams".

that there is no real need for the transport agent of the sending entity to wait for the acknowledgement of the previous DATA-TPDU before sending the next one.

This statement is the basis of the classical solution. As soon as it has some data ready to be transmitted, the transport agent of the sending entity would continuously transmit the corresponding DATA-TPDUs. On the other hand, the transport agent of the receiving entity would rebuild this data by looking at the sequence numbers of the incoming DATA-TPDUs.

Figure III.1 illustrates this solution. Entity A requests its transport agent to deliver the message Data to entity B. This message Data has to be split into three parts, say Data1, Data2 and Data3. These three parts are enclosed in DATA-TPDUs with appropriate sequence numbers, and these DATA-TPDUs are transmitted to the transport agent of B, using the network layer. For some obscure reason, the DATA-TPDU with sequence number 1 is delayed by the network layer, but the two remaining DATA-TPDUs are quickly delivered to their destinator[5]. The transport agent of entity B can still completely rebuild the message Data, and deliver the latter to entity B.

- Figure III.1 -
Dealing with datagrams re-ordering.



5. The possibility of such a phenomenon was discussed in paragraph I.7.

But this solution can not be applied "as-is":
- The acknowledgement scheme has to be reconsidered, for implementation and performance reasons.
- There is a need to keep the transport agent of the sending entity from transmitting too much DATA-TPDUs in a too short period of time.

*** The problem of acknowledgements.**

For the acknowledgement and retransmission on timeout mechanisms, the closest extension of the trivial solution to the new one would be this one:
- For each transmitted DATA-TPDU, an ACK-TPDU with the same sequence number is expected.
- As soon as this ACK-TPDU is received, the DATA-TPDU is acknowledged.
- If this ACK-TPDU is not received within some period of time, the DATA-TPDU is retransmitted and the corresponding ACK-TPDU is once again expected[6].

There is a problem of resource consumption. First, for each transmitted DATA-TPDU, a separate ACK-TPDU must travel on the network. This consumes a non-negligible part of the network bandwidth. But one can easily imagine a scheme where a single ACK-TPDU would acknowledge several DATA-TPDUs. This would lead to less network traffic. Second, for each transmitted DATA-TPDU, the transport agent of the sending entity must set up a timer which would start the retransmission procedure after the timeout period. In many operating systems, the number of such timers is a constraint.

There is also a problem of performance. It comes from timers handling by the operating system. Three basic operations are related to timers:
- a process sets up a timer,
- a timer expires and starts a process which executes a procedure,
- a process cancels a timer;

and in most operating systems, the fact is that the two first operations are quickly performed, but the last one consumes quite a lot processing time. And each time a DATA-TPDU is acknowledged, its associated timer must be cancelled! Thus expecting a separate acknowledgement for each transmitted DATA-TPDU consumes a non-negligible part of the CPU time, which could have been used to prepare the next DATA-TPDUs to be transmitted.

---

6. If this retransmission occurs too many times for the same DATA-TPDU, the transport agent may infer that there is a problem with the network layer, and thus can take some exceptional measures.

*** Cumulative acknowledgements.**

It is clear that each separate DATA-TPDU has to be actually acknowledged, even if no separate ACK-TPDU is generated by the transport agent of the receiving entity. So the semantic of the information included in an ACK-TPDU must change. In the trivial solution, the identifier of the ACK-TPDU was the sequence number of the DATA-TPDU it acknowledges[7]. In the classical solution, the identifier of the ACK-TPDU is a threshold sequence number; and all DATA-TPDUs with a sequence number less or equal to this threshold are acknowledged by this ACK-TPDU. This scheme is often referred to as the 'cumulative acknowledgement'.

It must be noted that, at a semantic point of view, the cumulative acknowledgement scheme is insensitive to the network layer problems discussed at paragraphs I.4 to I.7:

- When an ACK-TPDU is lost, the next one acknowledges at least the same DATA-TPDUs.
- When an ACK-TPDU is corrupted, it is discarded by the transport agent of the sending entity, and the next one acknowledges at least the same DATA-TPDUs.
- When an ACK-TPDU is duplicated, the extra copy acknowledges exactly the same DATA-TPDUs as the original one.
- When two consecutive ACK-TPDUs are swapped by the network layer, the second received ACK-TPDU acknowledges DATA-TPDUs that where already acknowledged by the first received one.

Many schemes can be imagined to choose the proper time for the transport agent of the receiving entity to issue an ACK-TPDU. Here comes some examples:

- every time a given number of DATA-TPDUs have been received since the latest ACK-TPDU was issued,
- when a given amount of time has elapsed since the latest DATA-TPDU was received[8],
- periodically,
- a composite of previous schemes.

*** Retransmission on timeout.**

Even when using the cumulative acknowledgement scheme, the retransmission on timeout mechanism could be used "as-is" in the classical solution. The only difference is that, on reception of an ACK-TPDU, several timers would sometimes have to be cancelled. But this scheme would lead to use a timer for each DATA-TPDU, a situation which must be avoided, as shown earlier.

---

7. This has been discussed in paragraph II.2, under the title "Dealing with duplicated datagrams".
8. This allows the transport agent of the receiving entity not to transmit ACK-TPDUs while receiving a burst of DATA-TPDUs.

An analysis of what would happen with this undesirable scheme when a DATA-TPDU is lost can help find a better solution. When this situation occurs, the timer associated with this DATA-TPDU expires, and the retransmission procedure starts. At this moment, none of the subsequent DATA-TPDUs are acknowledged, otherwise the lost DATA-TPDU would also be acknowledged. A problem comes from the conjunction of two facts:

- The acknowledgement for the retransmitted DATA-TPDU and the subsequent ones will come at least one RTD[9] later.
- Several DATA-TPDUs were also transmitted within the RTD following the first transmission of the lost DATA-TPDU.

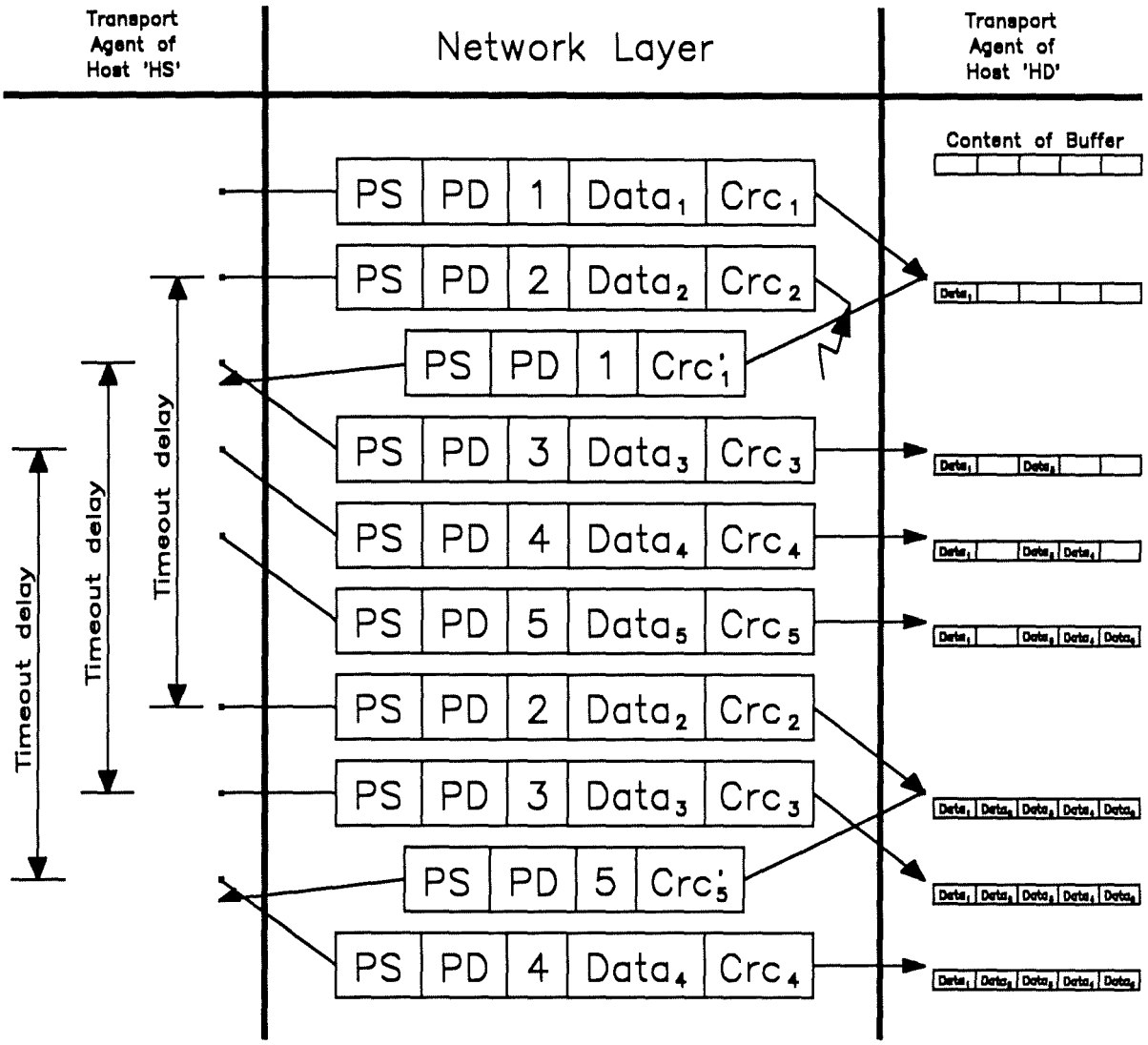
The consequence is that the timers associated with these DATA-TPDUs are very likely to expire, without any regard to their state at the transport agent of the receiving entity point of view.

Figure III.2 illustrates this analysis with a possible scenario for a transfer of data from entity A to entity B. The transport agent of A transmits the DATA-TPDUs with sequence numbers 1 to 5. The DATA-TPDU with sequence number 1 is correctly received by the transport agent of B, which responds a ACK-TPDU numbered 1. The DATA-TPDU with sequence number 2 is lost by the network layer and the DATA-TPDUs with sequence numbers 3 to 5 are well delivered to the transport agent of B. From this moment, no ACK-TPDU with a number higher than 1 can be transmitted, as the DATA-TPDU with sequence number 2 has not been delivered. After some period of time, the timer associated to the DATA-TPDU with a sequence number 2 expires, and this DATA-TPDU is retransmitted. This latest DATA-TPDU is finally delivered to the transport agent of B, which can then respond a ACK-TPDU numbered 5. But before this ACK-TPDU is delivered to the transport agent of A, the timers associated to DATA-TPDUs with sequence numbers 3 and 4 have also expired, and these DATA-TPDUs have been retransmitted.

There is an easy solution to get a similar behaviour with a single timer at the transport agent of the sending entity end. This single timer is to be associated to the non-acknowledged DATA-TPDU with the lowest sequence number. The transport agent of the sending entity reacts to timer expiration and to incoming ACK-TPDUs as described in figure III.3. The most interesting rule is the first one: when the timer expires, the transmission of DATA-TPDUs restarts from the first non-acknowledged one, for retransmitting it as well as all subsequent DATA-TPDUs. It is from this fact that this retransmission on timeout scheme is sometimes referred to by the expression 'go-back-N scheme'.

---

9. This has been discussed in paragraph II.2, under the title "Dealing with datagram loss".



- Figure III.2 -
The retransmission on timeout scheme
with a timer associated to each DATA-TPDU.

Figure III.4 illustrates this go-back-N scheme with the same initial scenario as in figure III.2. The transport agent of A transmits the DATA-TPDUs with sequence numbers 1 to 5. The DATA-TPDU with sequence number 1 is correctly received by the transport agent of B, which responds a ACK-TPDU numbered 1. The DATA-TPDU with sequence number 2 is lost by the network layer and the DATA-TPDUs with sequence numbers 3 to 5 are well delivered to the transport agent of B. From this moment, no ACK-TPDU with a number higher than 1 can be transmitted, as the DATA-TPDU with sequence number 2 has not been received. After some period of time, the timer expires, and so the transport

Event	Action to perform
Timer expires	Next <- Ack ; Reset timer ;
ACK-TPDU with number Num is received and Num <= Ack	Nothing to perform ;
ACK-TPDU with number Num is received and Ack < Num <= Next	Ack <- Num ; Reset timer ;
ACK-TPDU with number Num is received and Next < Num	Ack <- Num ; Next <- Num ; Reset timer ;

- Ack is the highest sequence number of the acknowledged DATA-TPDUs.
- Next is the sequence number of the next DATA-TPDU to be transmitted.

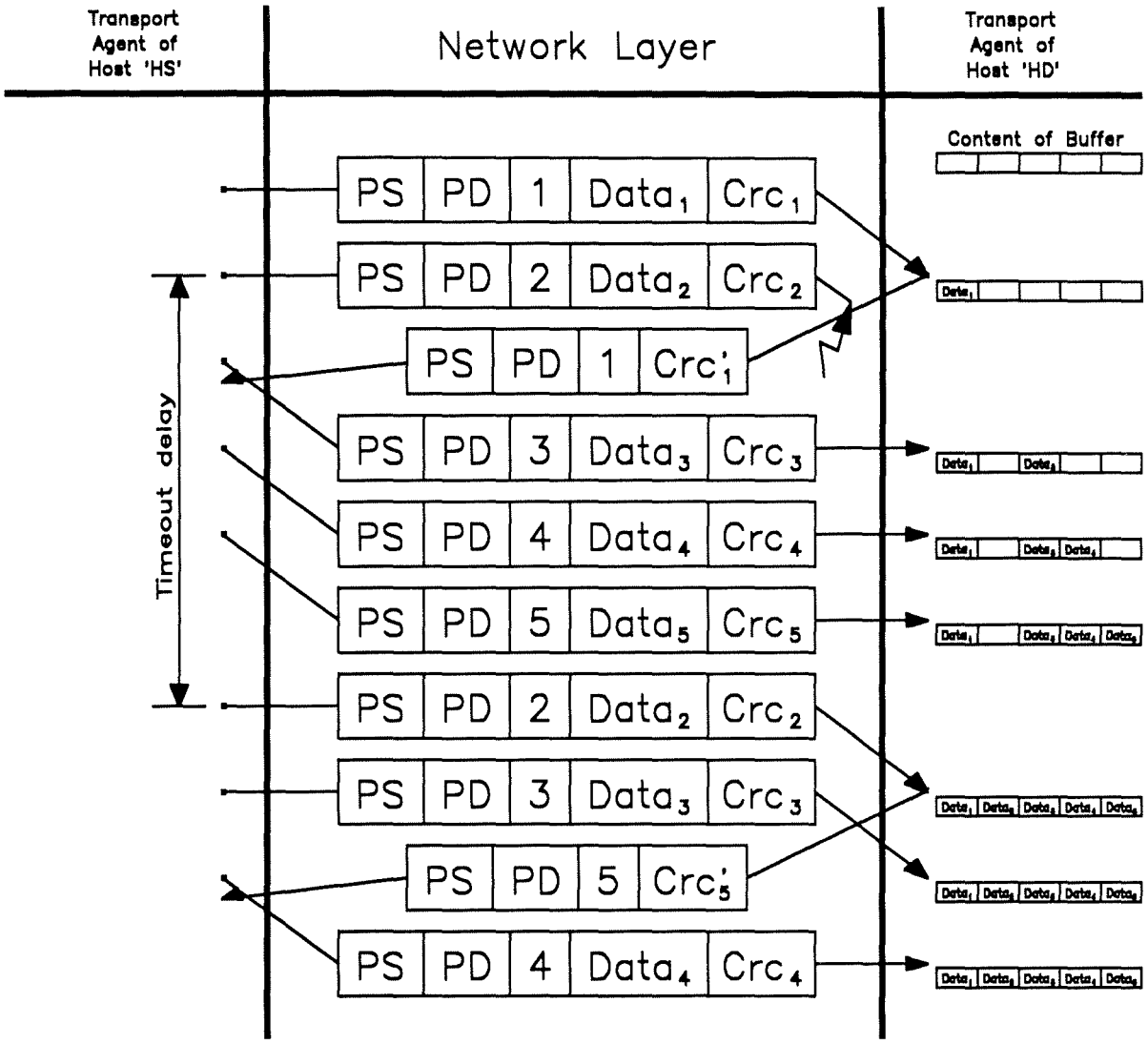
- Figure III.3 -
Rules of the go-back-N scheme.

agent of A retransmits DATA-TPDUs with sequence numbers 2 and higher. When the DATA-TPDU with sequence number 2 is delivered to the transport agent of B, an ACK-TPDU numbered 5 is transmitted. When the transport agent of A receives this ACK-TPDU, it stops its transmission as the next DATA-TPDU to be transmitted, the one with sequence number 6, is not yet ready. It is clear that the traffic of TPDUs is very similar to the one in the figure III.2.

An important question is the choice of the timeout delay to use for the timer. It is harder to compute than in the trivial solution case. Not only the RTD must be taken into account, but also the strategy the transport agent of the receiving entity uses for choosing when to transmit ACK-TPDUs. The difficulty is even worse when applying the third rule of figure III.3. In this case, there is a need to set up a timer associated to a DATA-TPDU that has been transmitted "some time ago".

*** The SOS region concept.**

Between the moment the transport agent of the sending entity transmits a DATA-TPDU and the moment it receives the corresponding acknowledgement, this DATA-TPDU is said to have its state 'out-of-synchronization'. This expression only means that during this period of time, the transport agent of the sending entity does not know the status of this DATA-TPDU at the destinating transport agent point of view.



- Figure III.4 -
The go-back-N scheme.

The set of all DATA-TPDUs which state is 'out-of-synchronization' is often referred to as the 'state out-of-synchronization region', abbreviated as the 'SOS region'.

In the trivial solution, we could have said that the SOS region was the unique DATA-TPDU which has been transmitted but has not yet been acknowledged. In the classical solution, this SOS region is far much larger: it consists of the sequence of all DATA-TPDUs which have been transmitted by the transport agent of the sending entity, and for which an acknowledgement is waited for. So this SOS region is identified by two numbers:

- the sequence number of the first transmitted-but-not-yet-acknowledged DATA-TPDU,
- the sequence number of the last transmitted-but-not-yet-acknowledged DATA-TPDU[10].

The SOS region length must be large enough to allow continuous transmission of DATA-TPDUs: this is the aim of the classical solution.

*** The transmission window.**

The length of the SOS region must be bound, as each DATA-TPDU in the SOS region consumes the memory resource of the transport agents of both sending and receiving entities:

- The data the transport agent of the sending entity includes in a DATA-TPDU must be remembered as long as this DATA-TPDU remains in the SOS region, as this data has to be included in all the retransmissions of this DATA-TPDU.
- The data a DATA-TPDU includes must be copied into the buffer space of the transport agent of the receiving entity until this entity 'consumes' this data.

There is no real memory problem at the transport agent of the sending entity point of view. Each data the sending entity delivers to its transport agent is stored in the memory, and this memory can be locked until the data it contains leaves the SOS region.

On the other hand, the available buffer space for the transport agent of the receiving entity to store incoming data is a real constraint. If the transport layer were allowing a DATA-TPDU to come to the transport agent of the receiving entity when there is no more space to store its data part, this data would have to be dropped and the DATA-TPDU retransmitted later.

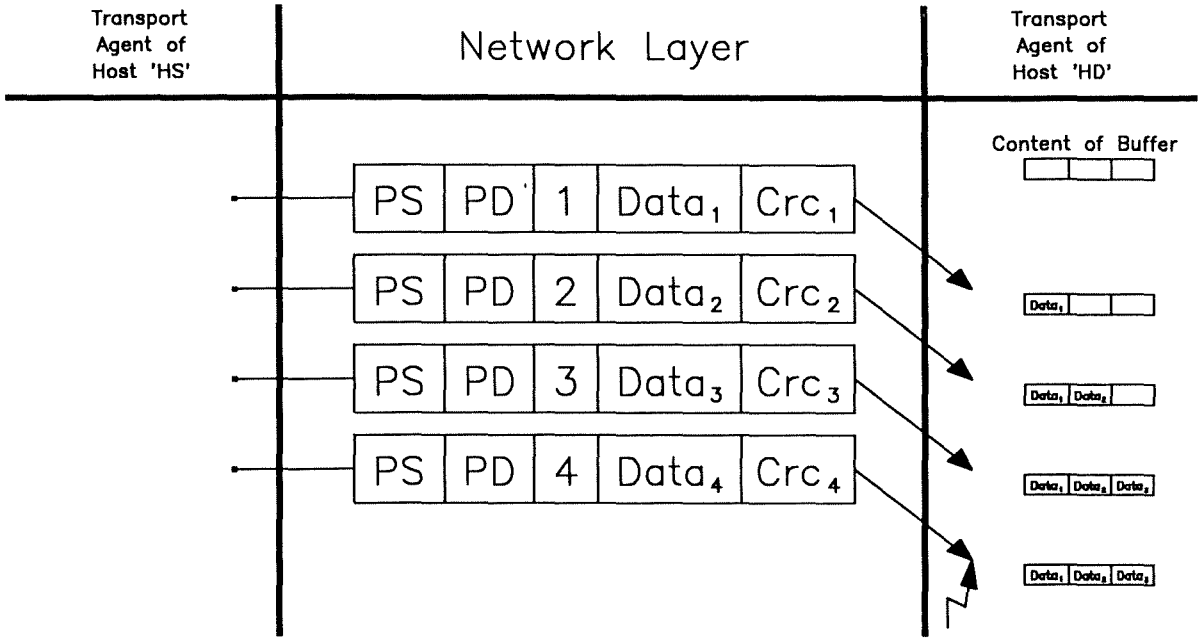
Such a situation is depicted in figure III.5. The transport agent of A transmits the DATA-TPDUs with sequence numbers from 1 to 4. But the transport agent of B has only got available buffer space for three DATA-TPDUs. So it can only store the data from DATA-TPDUs with sequence numbers from 1 to 3, and then it has to drop the DATA-TPDU with sequence number 4.

So there is a need to bind the length of the SOS region. This can be done by defining a 'transmission window' inside which the SOS region will be allowed to grow. This transmission window will be identified by two numbers: the sequence numbers of the first and the last DATA-TPDUs it contains[11].

---

[10]. Another equivalent solution is to identify the SOS region using the sequence number of its first DATA-TPDU and its length.

[11]. Another equivalent scheme to identify the transmission window is to use the sequence of its first DATA-TPDU and its length.



- Figure III.5 -
The loss of a DATA-TPDU due to a lack of buffering space.

*** Transmission authorizations.**

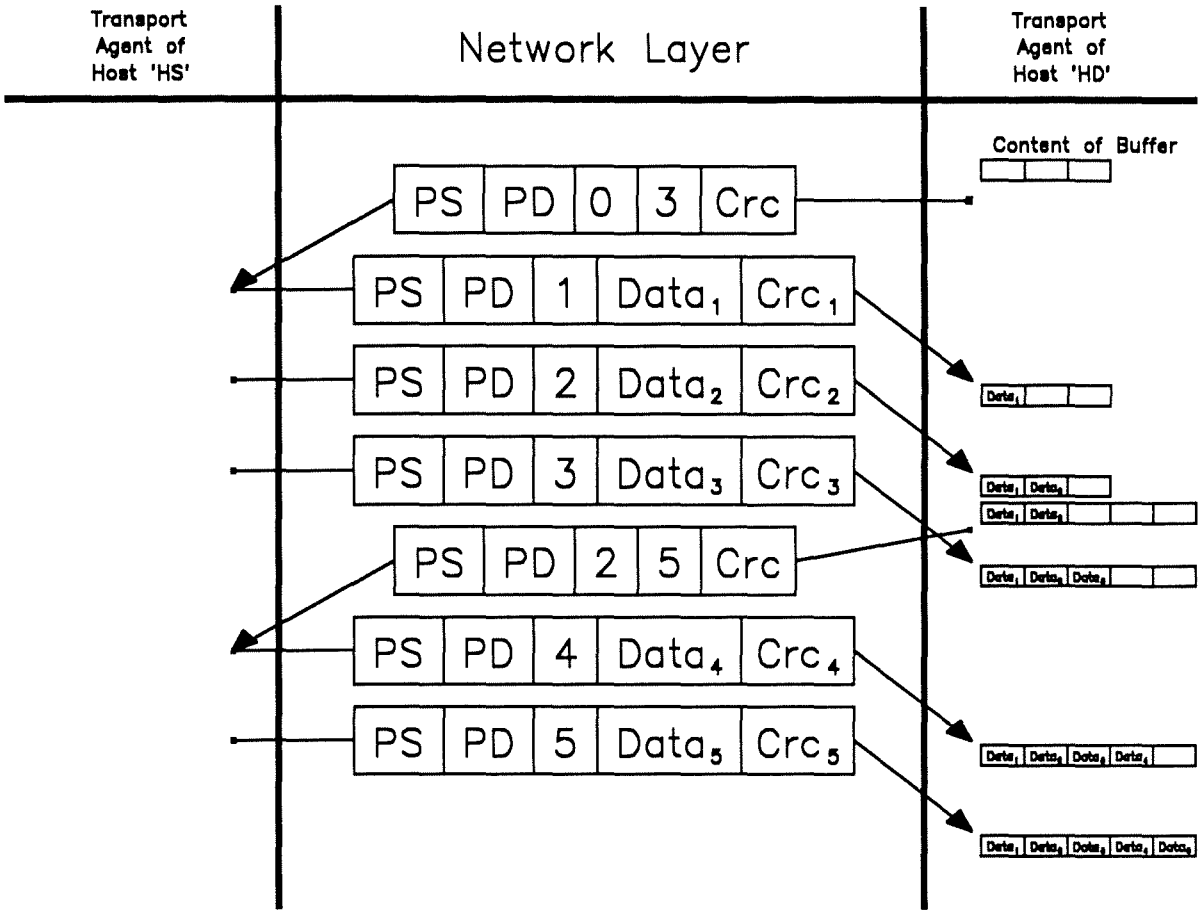
The major problem is that the available buffer space at the transport agent of the receiving entity end is an information the transport agent of the sending entity can not guess. So the former will have to feed back the latter with one more information: the highest DATA-TPDU sequence number it can accept. An interesting way to send this number is to include it in a new field of the ACK-TPDU.

As this number authorizes the transport agent of the sending entity to transmit DATA-TPDUs until a given sequence number, it is referred to by the expression 'transmission authorization'.

As this new ACK-TPDU carries more information than a bare acknowledgement, it will be referred to by the expression 'CONTROL-TPDU'.

Figure III.6 illustrates the transmission authorization scheme. First the transport agent of B warns the transport agent of A that:

- the DATA-TPDUs with sequence numbers lower or equal to 0 have been well received,
- there is an available buffer space for the DATA-TPDUs with sequence numbers lower or equal to 3.



- Figure III.6 -
The transmission authorization principle.

On reception of this CONTROL-TPDU, the transport agent of A starts transmitting DATA-TPDUs with sequence numbers form 1 to 3. After a period of time, the transport agent of B has got some more free buffering space for two DATA-TPDUs, and so it transmits a CONTROL-TPDU with a transmission authorization of 5. As at this moment, DATA-TPDUs with sequence numbers 1 and 2 were received, the acknowledgement value of this CONTROL-TPDU is 2. On reception of this TPDU, the transport agent of A starts transmitting DATA-TPDUs with sequence numbers form 4 to 5.

*** Recapitulation.**

There are problems of resource consumption and performance with a direct extension of the trivial solution with the transport agent of the sending entity continuously transmitting DATA-TPDUs. These have been solved by introducing the cumulative acknowledgements and go-back-N schemes.

There is no need to transmit DATA-TPDUs when the transport agent of the receiving entity has got no buffer space to store their data part. The transmission of such DATA-TPDUs is avoided by using the mechanisms of the transmission window and the window authorization.

III.3. Examples of the classical solution.

This classical solution is used by the standard transport protocol of the DOD world: the Transmission Control Protocol (TCP)[12]. This transport protocol is implemented on top of IP. Nowadays, almost all computers are sold with a version of TCP/IP.

The principles of this solution are also used by the Transport Protocol, class 4 (TP-4) of the connection-oriented OSI stack[13].

III.4. Results of the classical solution.

Many tests of implementations of TP4/CLNS and TCP/IP have been carried out on various equipments, ranging from personal computers to workstations and mainframes, connected to LANs[14]. The results of the throughput measurements of these examples of the classical solution are the basis on which the merits of the solutions described in the subsequent chapters will be discussed.

For the TP4/CLNS case, the measured throughput generally ranges from seven hundred kilobits per second to one megabit per second, over a ten megabits per second Ethernet or token bus local area network.

For the TCP/IP implementations, the results roughly vary from one to three megabits per second, over a ten megabits per second Ethernet local area network.

III.5. Analysis of the classical solution.

There are some problems with the classical solution, in case of TPDU loss. When a TPDU is lost, the retransmission timer

12. A complete description of TCP can be found in [RFC-793].
13. A complete description of TP-4 can be found in [ISO-8073].
14. The detailed results of these tests may be read in [PERFORMANCES].

expires after the timeout delay. At this moment, the transmission restarts from the first non-acknowledged DATA-TPDU. The next CONTROL-TPDU comes at least one RTD later. And, within this RTD time, DATA-TPDUs are retransmitted whether there were already correctly received or not. So for a single lost TPDU, there can be several unneeded DATA-TPDUs retransmission, leading to poor effectiveness.

This problem of effectiveness is also shown on figure III.4. height DATA-TPDUs have been transmitted by the transport agent of A. But in fact only six of them were really required for an effective transfer:

- the first transmission of the DATA-TPDUs with sequence numbers from 1 to 5,
- the retransmission of the DATA-TPDU with sequence number 2.

A numerical value of the effectiveness of this transfer can be computed as the ratio of the amount of required DATA-TPDUs by the amount of actually transmitted DATA-TPDUs. In this case, this effectiveness measurement would be 6/8, or 75%.

Two different kinds of effectiveness can be analysed:

- the local effectiveness, concerning the DATA-TPDUs inside the SOS region at the moment the TPDU is lost,
- the global effectiveness, concerning the transfer of the entire data.

The local effectiveness is mainly function of the rate of DATA-TPDUs emission and the RTD value. The minimum amount of retransmitted DATA-TPDUs is in fact the product of these two values. The global effectiveness will be mainly function of the rate of TPDU losses and the local effectivenesses when these TPDUs are lost.

It is interesting to analyse the origins of TPDUs loss, and the impact of the different classes of loss on the global effectiveness of the transfer. Two reasons for loosing TPDUs were already encountered in the trivial solution: first, the network layer sometimes drops a TPDU[15], when one of its routers in congested; second, a transport agent discards a TPDU when the latter fails the integrity checking[16]. But in the classical solution, a third reason for loosing TPDUs appears, as the result of three facts:

- A transport agent may continuously receive TPDUs.
- Each TPDU requires some processing time to be treated.
- Processing time is a finite resource,

These lead to a threshold on the rate of TPDU treatment. And when a transport agent receives TPDUs faster than it can treat them, it has nothing else to do than dropping some of them. An

15. This has been discussed in the paragraph I.4.
16. This has been discussed in paragraph II.2, under the title "Dealing with datagram corruption".

example of such a situation would be materialised by an entity running on a very fast host and transferring data to an entity running on a very slow host.

When the loss of TPDUs comes from the discarding of a corrupted TPDU, there is no real problem, as this can be considered as an unusual event. It leads to a momentary poor effectiveness, but on the long term, it has no impact on the global effectiveness of the data transfer.

The true problem comes when TPDUs are lost due to congestion of one router of the network layer, or to the congestion of the transport agent of the receiving entity. In this case, the transport layer reacts to the congestion by restarting the data transmission from the failing DATA-TPDU. But this does not solve the congestion problem in any way, and it is very likely that another TPDU will soon be lost due to congestion, and so on, and so on: this is an endless loop. And thus all along the time, the global effectiveness of the transfer will stay low.

Dr. Van Jacobsen has imagined a solution to this problem. In his scheme, the transport agent of the sending entity does not retransmit the entire transmission window after a timeout, but instead it only retransmits the first failing DATA-TPDU. When this DATA-TPDU gets acknowledged, the two next unacknowledged DATA-TPDUs are retransmitted, then a group of four DATA-TPDUs, and so on until the entire transmission window can be transmitted at once. This scheme is known as the 'slow-start algorithm'[17]. The name of this scheme comes from the fact that it is also interesting to use it for the very first DATA-TPDUs of the data transfer.

The conclusion of this analysis is that there is no way to achieve an optimal end-to-end throughput with the classical solution. If there is no congestion resolution scheme present, all the network bandwidth can be used, but sometimes with poor efficiency. On the other hand, all simple congestion avoidance schemes will be similar to Dr. Jacobsen's one, in the sense that they will keep the transmission window small, thus preventing the transport agent of the sending entity from continuously transmitting DATA-TPDUs. Thus a simple congestion avoidance scheme will lead to a waste some part of the network bandwidth.

III.6. Conclusion.

The classical solution meets its requirements, in the sense that it is a simple solution to improve the throughput of the trivial solution. Except for the transmission window one, all concepts are only evolutions of those used in the trivial

17. The interested reader may refer to [TCP-2] for a complete description of the slow-start algorithm.

solution. These two characteristics of simplicity and good throughput have lead this solution to be the currently most used one, with TCP and TP-4.

The further problem is that this classical solution only offers a good throughput, but not an optimal throughput. With a congestion resolution scheme, some resources are wasted because they are not used; and without such a scheme, some resources are also wasted because they are mis-used.

So there is a need to find another solution for the transport layer design, which use all the available resources, but in a more effective way.

Chapter IV:

An improved design.

IV.1. Aims of the improved design.

The aim of this improved design is to enhance the classical solution, in a way to allow a better end-to-end[1] throughput. This will be achieved by meeting both following goals:

- to have a high transmission rate between the transport agents of the sending and the receiving entities,
- to have a high global effectiveness in this transmission.

As shown in its analysis, the classical solution can only meet one of these goals at a time. It is very uncommon for it to meet both of them, especially when the network latency is long, when the network is heavy loaded, or when this network frequently corrupts datagrams[2].

The high transmission rate between both transport agents will be achieved by:

- never stopping the transport agent of the sending entity transmitting DATA-TPDUs before it has reached the end of its transmission window,
- making the transmission authorizations reflecting only buffering capacities of the transport agent of the receiving entity.

From the analysis of the classical solution, it is rather manifest that the high global effectiveness requirement will be achieved by:

- upgrading the local effectivity when a DATA-TPDU is lost,
- keeping low the rate of DATA-TPDU losses.

---

1. In other words: entity-to-entity.
2. See paragraph III.5.

IV.2. Principles of the improved design.[3]

* The selective acknowledgement mechanism.

In the go-back-N scheme, the problem of poor local effectiveness when a DATA-TPDU is lost is due to the unneeded retransmission of several DATA-TPDUs. This comes from the fact that, with a cumulative acknowledgement, the transport agent of the receiving entity does not send any information to its peer about the state at its end of the DATA-TPDUs inside the "non-acknowledged part" of the transmission window.

On the very opposite, one could imagine a scheme where each CONTROL-TPDU the transport agent of the receiving entity transmits to its peer carries a list of all DATA-TPDUs it has received so far. This acknowledgement mechanism is referred to as the 'selective acknowledgement' mechanism.

One problem with the selective acknowledgement scheme is the choice of the implementation of this list of received DATA-TPDUs. This implementation must lead to:

- a concise representation of the list, for an effective use of the available network bandwidth,
- an easy handling of this list by both transport agents, for an effective use of the available computing resource.

Two good compromises for such an implementation are:

- the list of all ranges of the sequence numbers of received DATA-TPDUs,
- the sequence number of the first missing DATA-TPDU, the sequence number of the last received DATA-TPDU, and a bitmap describing the state of all DATA-TPDUs within the range defined by this pair of sequence numbers.

But there are other more annoying problems with this scheme. The first one is the choice, for the transport agent of the receiving entity, of the right moment to transmit the CONTROL-TPDU carrying the selective acknowledgement. And the second one is the choice, for the transport agent of the sending entity, of the right moment to retransmit a non-acknowledged DATA-TPDU.

* The inadequacy of the retransmission on timeout scheme.

With a separate timer associated to each transmitted DATA-TPDU, at the transport agent of the sending entity end, a retransmission on timeout scheme may be handled with ease. The timeout delay would be chosen in respect to the current estimation of the RTD, as well as to the scheme used for the CONTROL-TPDUs transmission decision. But this solution is

---

[3]. A complete description of the improved design may be read in [NETBLT].

totally inadequate, as handling a separate timer for each transmitted DATA-TPDU would consume too much processing time on the host running the transport agent of the sending entity[4].

It is possible to imagine an extension of the go-back-N scheme which would use selective acknowledgements. A timer would be associated to the first non-acknowledged DATA-TPDU, and all the rules described in figure III.4 would be used[5], with the only difference that the process responsible for the transmission of the DATA-TPDUs would skip all acknowledged ones. But this scheme mainly leads to two annoying problems.

- First, the transport agent of the receiving entity regularly transmits CONTROL-TPDUs, which have a respectable size. This can consume a non negligible part of the network bandwidth.
- Second, when applying the third rule of figure III.4, it is very hard to compute an appropriate new timeout delay, as it depends on the moment a previous DATA-TPDU was transmitted.

So, for using a selective acknowledgement scheme, there is a need for other retransmission mechanisms than the classic retransmission on timeout one.

*** The periodic resynchronization scheme.**

An interesting substitute solution is to alternate periods of DATA-TPDU transmission and periods of state resynchronization. When using this scheme, it is up to the transport agent of the sending entity to take all decisions involving this state resynchronization.

First, the transport agent of the sending entity chooses some point in the data stream to be transmitted. Such a point is called a 'resynchronization point'. Then DATA-TPDUs are transmitted until this point is reached. But during this period, no acknowledgement at all is generated, and so no DATA-TPDU retransmission is carried out.

When the resynchronization point is reached, the DATA-TPDU transmission stops and the resynchronization procedure is performed. The latter consists of the transport agent of the sending entity to:

- ask the transport agent of the receiving entity for an acknowledgement,
- to retransmit all the non-acknowledged DATA-TPDUs,
- to repeat the two preceding steps until all DATA-TPDUs before the resynchronization point are acknowledged.

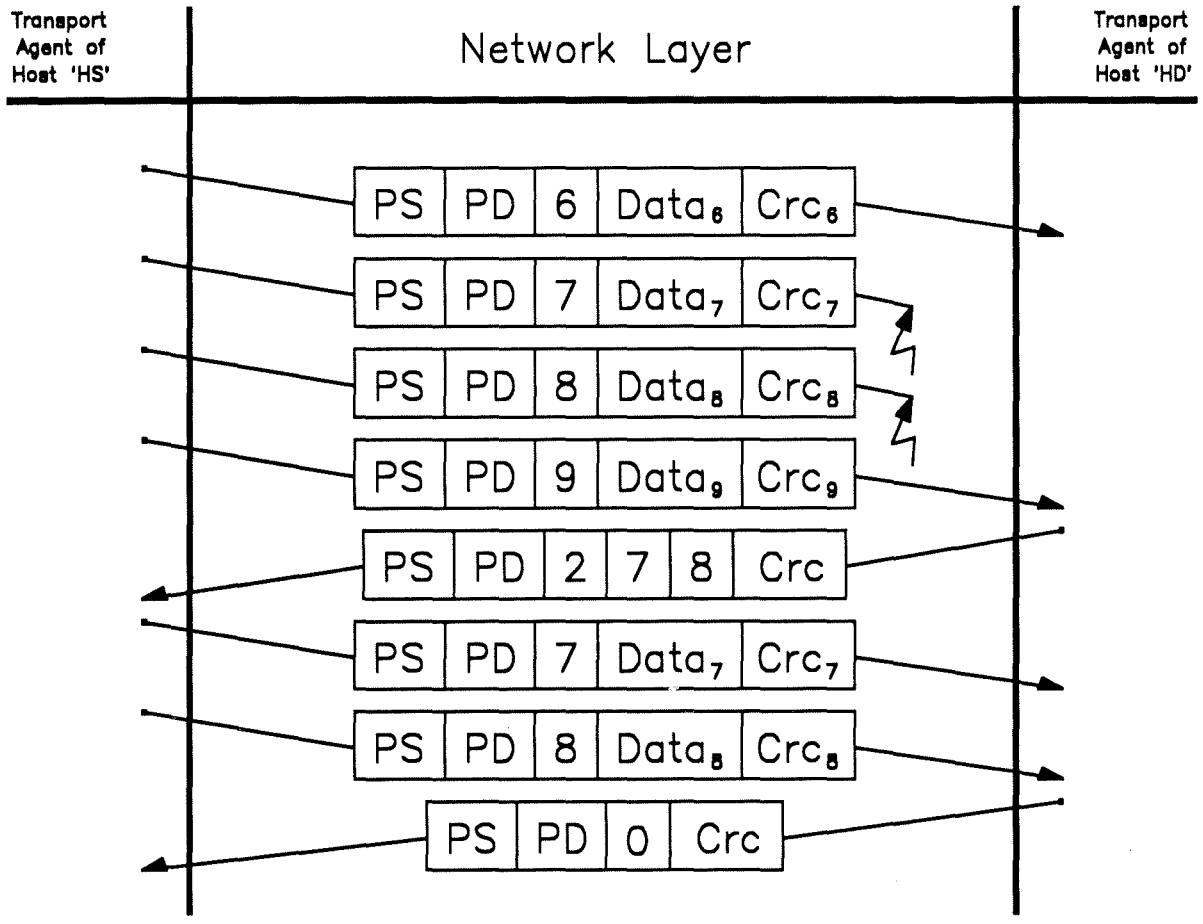
---

4. Such a problem was already encountered in paragraph III.2.
5. With Num computed from the content of the CONTROL-TPDU.

When both transport agents are resynchronized, the transport agent of the sending entity chooses the next resynchronization point, and the next DATA-TPDU transmission period starts.

Figure IV.1 illustrates this mechanism. The transport agent of entity on port 'ps' of host 'hs' transmits a message to the transport agent of entity on port 'pd' of host 'hd'. This message has to be split into four DATA-TPDUs, which have sequence numbers from 6 to 9. DATA-TPDUs with sequence numbers 6 and 9 are correctly transmitted, but DATA-TPDUs with sequence numbers 7 and 8 are lost by the network layer. A request for resynchronization is associated to the DATA-TPDU with sequence number 9. So, on reception of this DATA-TPDU, the transport agent on host 'hd' transmits a CONTROL-TPDU mentioning, among other things, the list of all DATA-TPDUs to be retransmitted (that is to say, those with sequence numbers 7 and 8). So the transport agent of host 'hs' retransmits both missing DATA-TPDUs, with an other request for retransmission associated to

- Figure IV.1 -
The periodic resynchronization scheme.



the DATA-TPDU with sequence number 8. Then a CONTROL-TPDU stating that all DATA-TPDUs have been correctly received is transmitted by the transport agent of host 'hd'.

The step where the transport agent of the sending entity asks its peer an acknowledgement is worth a detailed analysis. The fact that the transport agent of the sending entity requests an acknowledgement is an information that must be transferred, in a TPDU, to the transport agent of the receiving entity[6]. The problem is that this TPDU is subject to all network problems described in paragraphs I.4 to I.7, and so the principles of the trivial solution[7] must be applied. In fact the TPDU carrying the request for an acknowledgement will have to be retransmitted on timeout until the acknowledgement itself is received[8].

An important issue is the choice of the synchronization point. As a matter of fact, when the transport agent of the sending entity reaches such a point in the strict application of the above scheme, there is no more new DATA-TPDUs transmitted before all preceding DATA-TPDUs are acknowledged. So it can be interesting to choose as a synchronization point a 'natural' one: that is to say a point where the transport agent of the sending entity would have stopped transmitting DATA-TPDUs. There are two such natural points:

- the end of the transmission window,
- the end of a user message[9].

But in fact, there is no real reason to stop the DATA-TPDU transmission while a resynchronization takes place. And so when a resynchronization point is reached, the transport agent of the sending entity may perform two simultaneous tasks:

- to resynchronize all DATA-TPDU preceding the resynchronization point,
- to transmit DATA-TPDUs until the next resynchronization point.

And with this new evolution of the solution, some more arbitrary resynchronization points may also be chosen.

*** The retransmission on demand scheme.**

Another interesting substitute solution is often referred to as the 'retransmission on demand' scheme. Here are its principles.

The transmitted sequence of DATA-TPDUs is no more considered by both transport agents as a simple sequence of DATA-TPDUs, but

---

6. This could be done by raising a flag, say <RESYNC>, in the Flags fields of the last transmitted DATA-TPDU; or by issuing some king of special purpose packet, say RESYNC-TPDU.
7. They have been discussed in paragraph II.2.
8. After a given number of unsuccessful retransmissions, the transport agent of the sending entity may decide that there is a problem with the network layer and may start an exceptional procedure.
9. Assuming that there is often a significant delay between the deliveries of two successive user messages to the transport layer, for a single data stream.

as a sequence of blocks of DATA-TPDUs. This is implemented with a new identifying scheme for the DATA-TPDUs, where each DATA-TPDU is identified by the two following numbers:

- the sequence number of the block the DATA-TPDU belongs to,
- the sequence number of the DATA-TPDU, relative to the block it belongs to.

So, in each transmitted DATA-TPDU, the former sequence number field is replaced by two new fields which respective values are those just described. Furthermore, for the transport agent of the receiving entity to exactly know the content of each transmitted block, all DATA-TPDUs also indicate in one of their fields the number of DATA-TPDUs their block is composed of.

In its segmentation of the transmitted data stream into blocks, the transport agent of the sending entity always ensures that it will be able to transmit all the DATA-TPDUs of any block in a single burst. A good policy which conforms to this constraint is to separate each user messages from the others. So a single user message could be cut into a sequence of fixed length blocks and a last block for the remaining part of the message.

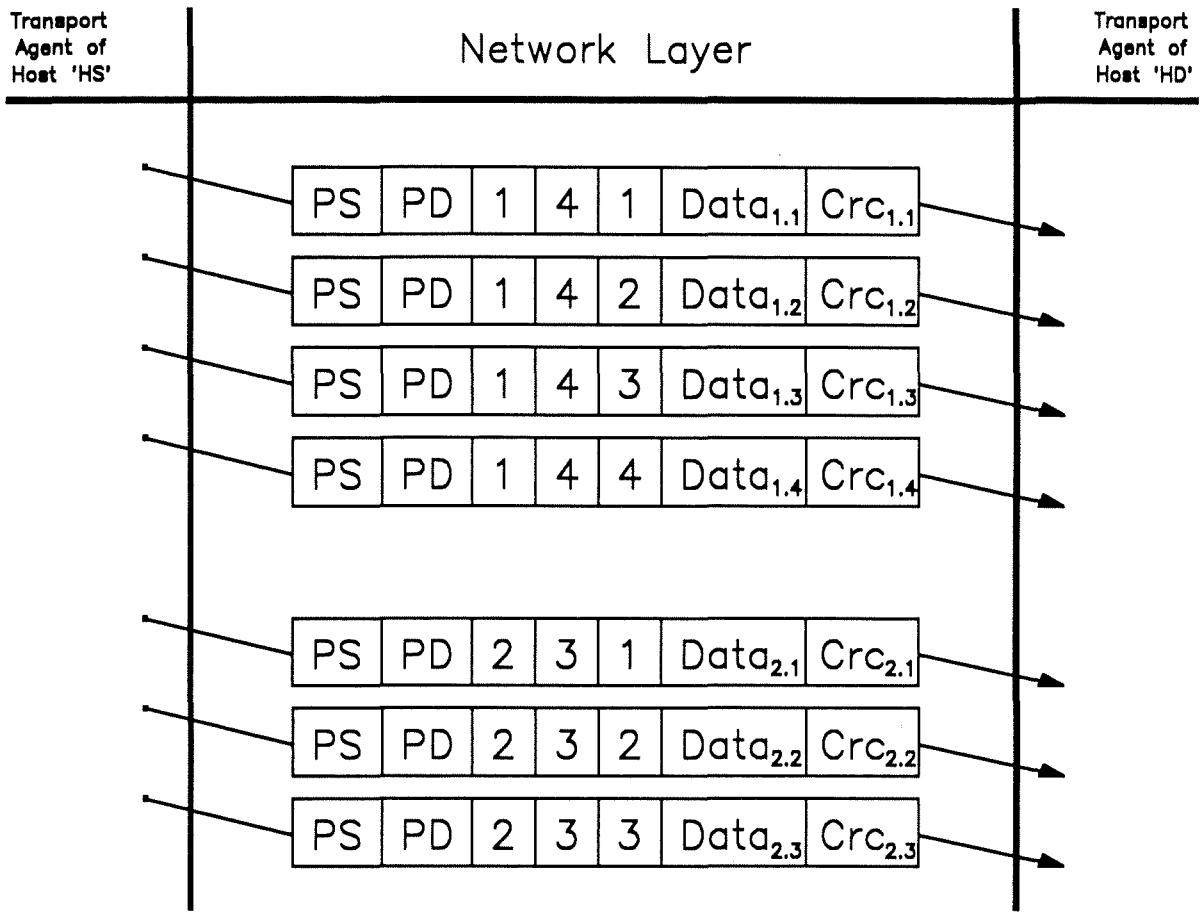
Figure IV.2 shows an example of this kind of packetization. Entity A delivered its transport agent a message which is destinated to entity B. This message is to be packetized into seven DATA-TPDUs, and the protocol specifies the maximal block size to be four DATA-TPDUs. So the message to be transmitted is divided into two blocks: the first block contains the first four DATA-TPDUs, and the second block the three remaining ones. On this diagram, the fields of the DATA-TPDUs are expressed in the following order:

- the source and destination port fields,
- the block sequence number field,
- the block size field,
- the DATA-TPDU sequence number field, relative to the block,
- the data field,
- the checksum field.

The detection of lost DATA-TPDUs can then be performed by the transport agent of the receiving entity. The latter can be aware of the fact that no DATA-TPDU within a block is lost when it receives the last missing DATA-TPDU that belongs to this block. On the other hand, it is also able to detect that a DATA-TPDU is lost within a block when the two next conditions meet:

- it has no yet received that DATA-TPDU,
- it has not received any DATA-TPDU of that block for 'a long time'.

This comes from the fact that the transport agent of the sending entity transmits all DATA-TPDUs of a block in a single burst: so all these DATA-TPDUs are due to be received by the transport agent of the receiving entity in a 'short' period of time. In



- Figure IV.2 -
The packetization of a user message
into blocks of DATA-TPDUs.

fact, an adequate value for the 'long time' mentioned above can be estimated in regard to the following statement: the preceding received DATA-TPDU of that block may have been transmitted by the network layer with a short latency, and the expected one may have been transmitted with a long latency. So this 'long time' has only to take into account the estimation of the variations in the network latency. And, as it had to be done in the RTD case, this estimation is to be based on measures of preceding values of these variations.

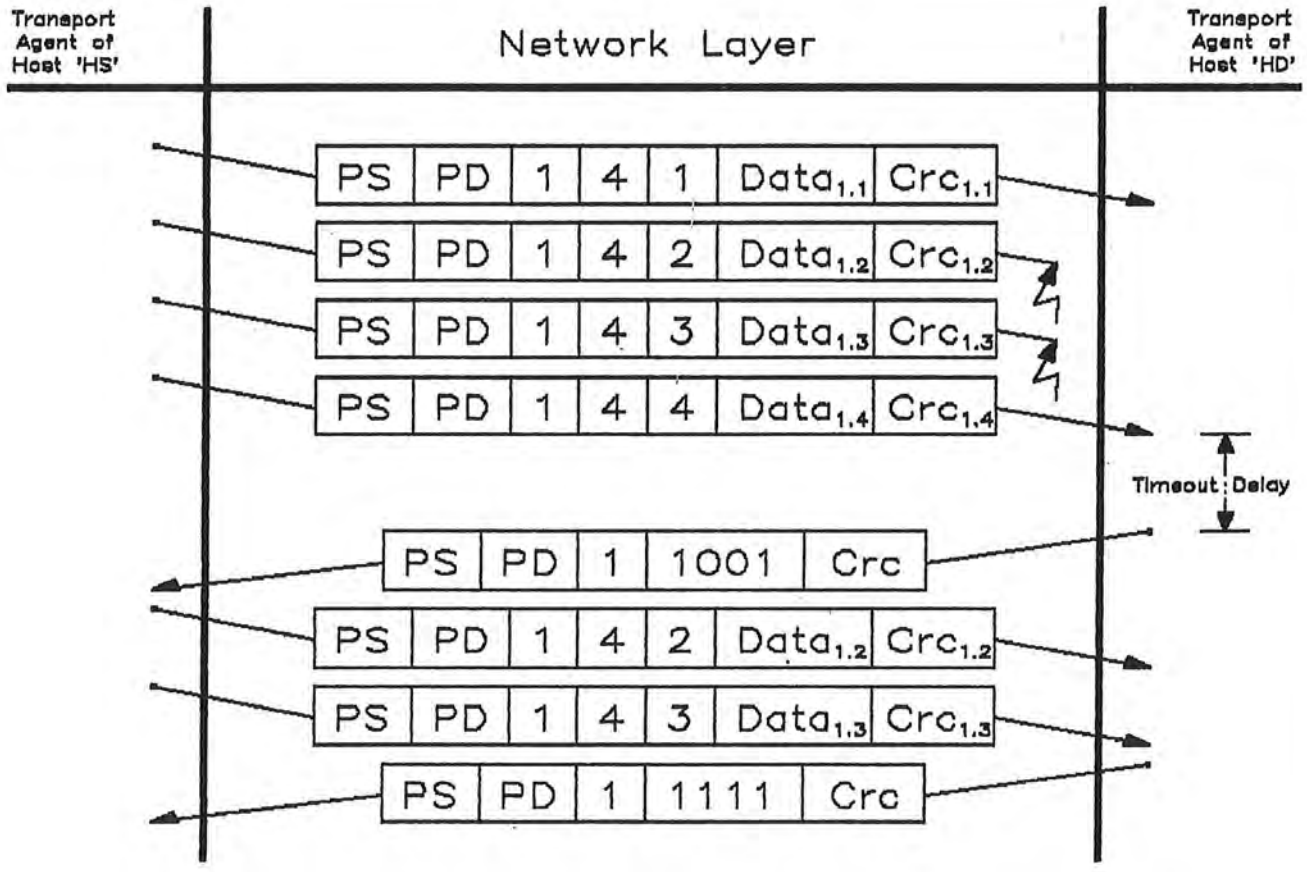
When all DATA-TPDUs of a block are received, or when the timer associated to this block expires, the transport agent of the receiving entity transmits a selective acknowledgement for that block (not for the entire stream). At the transport agent of the sending entity end, when a selective acknowledgement for a block is received, all DATA-TPDUs marked as missing in that block are retransmitted, in a single blast. The same algorithm

Mémoire S. SCHAECK 1984-90

Erratum.

The last modification of the pagination of this text has lead to an error on page 49. The printed figure is not figure IV.3 but another copy of figure IV.2. Here is the correct one.

- Figure IV.3 -
The retransmission on demand scheme.

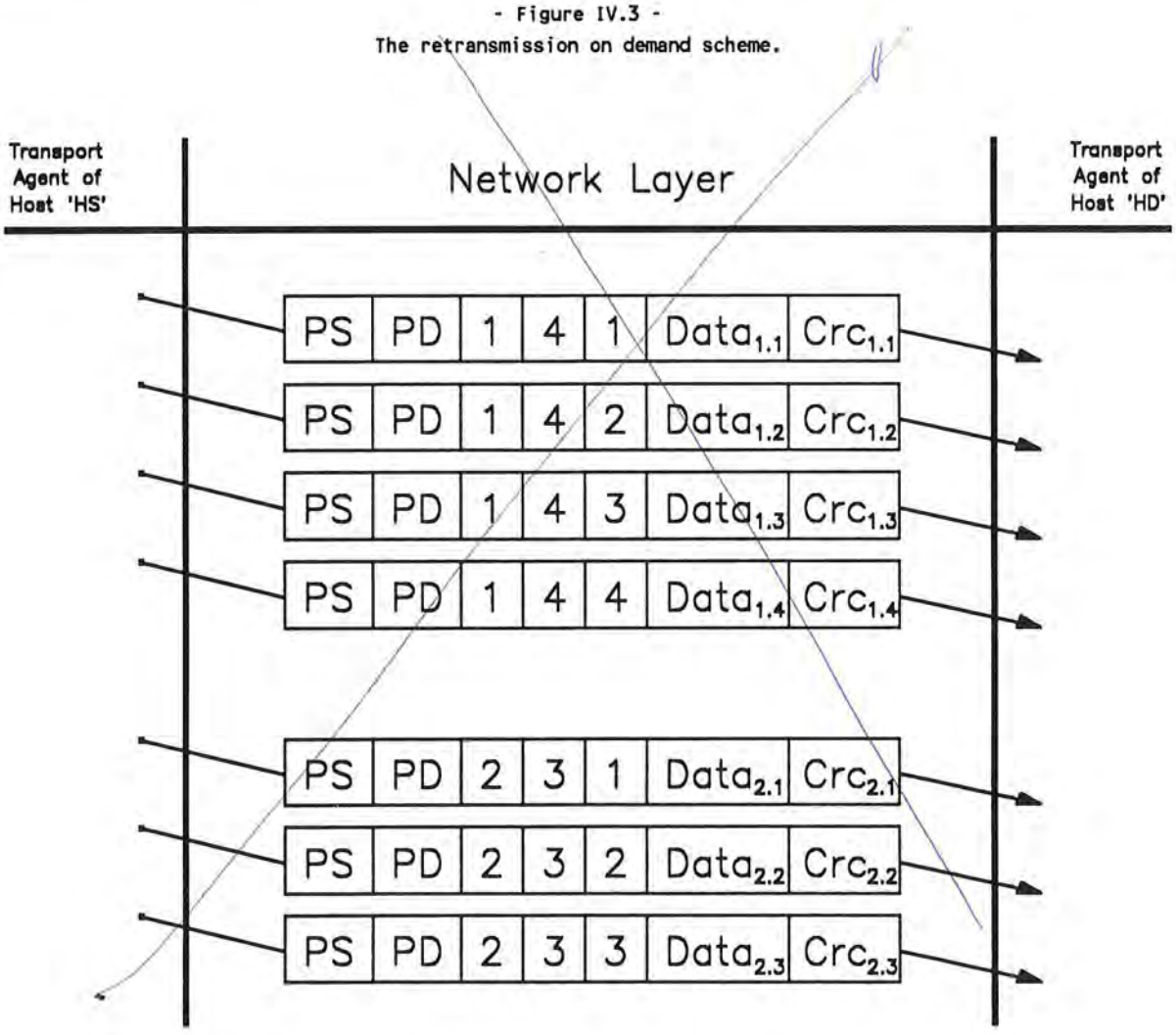


may be repetitively used to detect DATA-TPDUs which get lost during a retransmission and to request once again their retransmission, until all DATA-TPDUs of that block are correctly received.

The fact that the transport agent of the sending entity immediately retransmits missing DATA-TPDUs on reception of a selective acknowledgement can be viewed as the response to a request for the retransmission of these DATA-TPDUs. This is the reason why this scheme is referred to as the 'retransmission on demand' principle.

This retransmission on demand principle is illustrated by figure IV.3. For the communication between entity on port 'ps' of host 'hs' and the entity on port 'pd' of host 'hd', the transport agent of host 'hs' has just received a transmission authorization for the block with sequence number one. This block consists of four DATA-TPDUs, with relative sequence

- Figure IV.3 -
The retransmission on demand scheme.



numbers from one to four. On the first transmission of the entire block, only DATA-TPDUs with relative sequence number one and four are received, and the two other DATA-TPDUs are lost by the network layer. After some delay, the timer associated to this block at the receiving end expires, and so the transport agent of host 'hd' transmits a CONTROL-TPDU carrying a request for the retransmission of DATA-TPDUs two and three from the block with sequence number one. On reception of this TPDU, the transport agent of 'hs' immediately retransmit these two DATA-TPDUs, which this time are correctly delivered to the transport agent of host 'hd'. As soon as the latter has received these DATA-TPDUs, it transmits a CONTROL-TPDU carrying the acknowledgement for the entire block with sequence number one. For the CONTROL-TPDUs of this figure, only their acknowledging part is explicitly represented: a first field indicates which block the CONTROL-TPDU acknowledges, and a second one is a bitmap reflecting which DATA-TPDUs of that block were actually received.

This scheme works very well when at least one DATA-TPDU of every block is received by the transport agent of the receiving entity. But there is a problem if no DATA-TPDU of a given block is received. In this case, the transport agent of the receiving entity does not even know that the DATA-TPDUs of that block were transmitted, and so it is not able to transmit any demand for the retransmission of the entire block. This problem is solved by using the retransmission on timeout scheme at the block level: if, some time after the last DATA-TPDU of a block is transmitted, no acknowledgement for that block is received, the transport agent of the sending entity retransmits the entire block.

But there is an other problem coming with the above solution, as one of the following statements is true when the timer for the retransmission on timeout of the entire block expires:

- No DATA-TPDU of that block was received during the preceding transmission of the block.
- The selective acknowledgement the transport agent of the receiving entity delivered to the network layer got lost.

It is rather evident that the probability of the loss of the single TPDU carrying the acknowledgement is far much higher than the probability of the loss of all DATA-TPDUs used to transmit the block[10]. The worse case arises from the loss of an acknowledgement requesting no retransmission at all: the entire block has been correctly received, but it is entirely retransmitted. The conclusion is that the retransmission on demand scheme is very sensitive to the loss of CONTROL-TPDUs.

10. Assuming that a block consists of several DATA-TPDUs.

So, to meet the requirements of the improved design[11], there is a need for implementing supplementary mechanisms alongside the retransmission on demand scheme:

- to enhance the chance of delivery of CONTROL-TPDUs,
- to moderate the unneeded retransmissions in case of CONTROL-TPDU loss.

*** The flow control between transport agents.**

The above mechanisms dealt with upgrading the local effectiveness of the TPDUs exchange in case of DATA-TPDU loss. The second objective is to lower the rate of TPDUs loss.

It has been shown that a high rate of TPDUs loss is caused by some congestion[12]:

- of the route between both transport agents,
- of a transport agent itself.

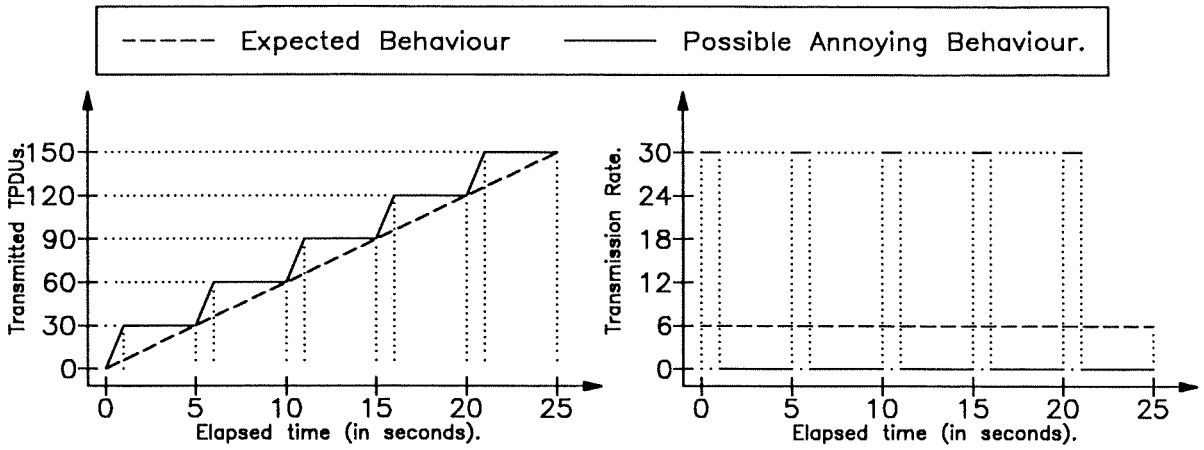
This congestion occurs when a transport agent transmits TPDUs faster than a threshold rate, which can be referred to as the 'congestion rate'. This congestion rate can be defined as the maximal rate at which the network and the peer transport agent can treat DATA-TPDUs. In fact, the problem of lowering the rate of TPDUs loss is a problem of congestion avoidance, and this problem of congestion avoidance is a problem of flow control between transport agents: the transport agent of the sending entity must be required not to transmit DATA-TPDUs with a higher rate than the congestion one.

In the classical solution, the only mechanism limiting the transmission of TPDUs is the transmission window[13] one. But it is not sufficient: a transmission window only binds the amount of DATA-TPDUs to be transmitted, and not the rate at which they can be transmitted. It can be objected that the transmission authorizations are received by the transport agent of the sending entity with a given rate, and so that this rate of window authorization reception indirectly limits the transmission rate. This deduction is quite approximate, and to be more accurate, the transmission authorizations bind the average rate of DATA-TPDUs transmission, but not any instant rate. And nothing prevents from pitches in the transmission rate, with instant transmissions rates far higher than the congestion one.

An illustration of this problem is depicted on figure IV.4. A transport agent receives, every five seconds, a transmission authorization for thirty DATA-TPDUs. But nothing prevents it from transmitting these thirty DATA-TPDUs within a single second and than waiting for four seconds. The left graph shows the progression of the amount of transmitted DATA-TPDUs, in function

---

11. These requirements have been discussed in paragraph IV.1.
12. This has been discussed in paragraph III.5.
13. This has been described in paragraph III.2.



- Figure IV.4 -
The transmission rate control using window authorizations.

of the elapsed time. The right graph shows the instant transmission rate, in function of the elapsed time. In both graph, dashed lines represent what could have been expected, and solid lines represent what can really happen.

A primitive aspirant as a solution to the above problem is based on the following consideration: an instant transmission rate is nothing else than an average transmission rate calculated for an infinitesimal period of time[14]. So, a solution, for the transport agent of the receiving entity to control the flow of DATA-TPDUs its peer transmits, would be to send shorter transmission authorizations at a higher rate.

But this solution is not satisfying for two reasons. First, it would boost the traffic of CONTROL-TPDUs. This interferes with all the efforts made to lower this traffic, which consumes a considerable part of the available network bandwidth for a non-effective purpose. Second, it would only control the flow between transport agents for the first transmission of the DATA-TPDUs, and would not help during the retransmission of missing ones. As a matter of fact, the transmission authorizations only deal with the DATA-TPDUs to be transmitted for the first time, and all the ones to be retransmitted come as a supplement. This is sometimes referred to as an 'out-of-band retransmission'.

14. The instant transmission rate can be viewed as the derivative of the amount of transmitted DATA-TPDUs, function of the elapsed time.

*** The rate-based flow-control.**

The primitive solution described above indicates at least one requirement for a good flow control between transport agents: the transport agent of the sending entity must explicitly use a rate control mechanism when transmitting DATA-TPDUs. Furthermore, it must transmit all DATA-TPDUs using the chosen rate, whether these DATA-TPDUs are transmitted for the first time or not. This is sometimes referred to as the 'in-band retransmission' mechanism.

Now, as the transport agent of the sending entity explicitly controls its transmission rate, the transport agent of the receiving entity can have an easy control on this transmission rate. It can do so by only telling its peer the rate at which it wants the DATA-TPDUs to be transmitted. This information can be carried in a new field of the CONTROL-TPDUs the transport agent of the receiving entity transmits to its peer[15]. At the other end, the transport agent of the sending entity would choose as its transmission rate the one mentioned in the last CONTROL-TPDU it received.

The next question for the transport agent of the receiving entity is then: what is the appropriate transmission rate to expect? The theoretical response is: a rate which is lower than the congestion rate, but very close to it. But the problem is that the transport agent of the receiving entity does not know the value of this congestion rate. A good practical response to the question is to dynamically adjust the transmission rate using the two following rules:

- when DATA-TPDUs are regularly lost, decrease the transmission rate,
- if no DATA-TPDU has been lost for a while, increase the transmission rate.

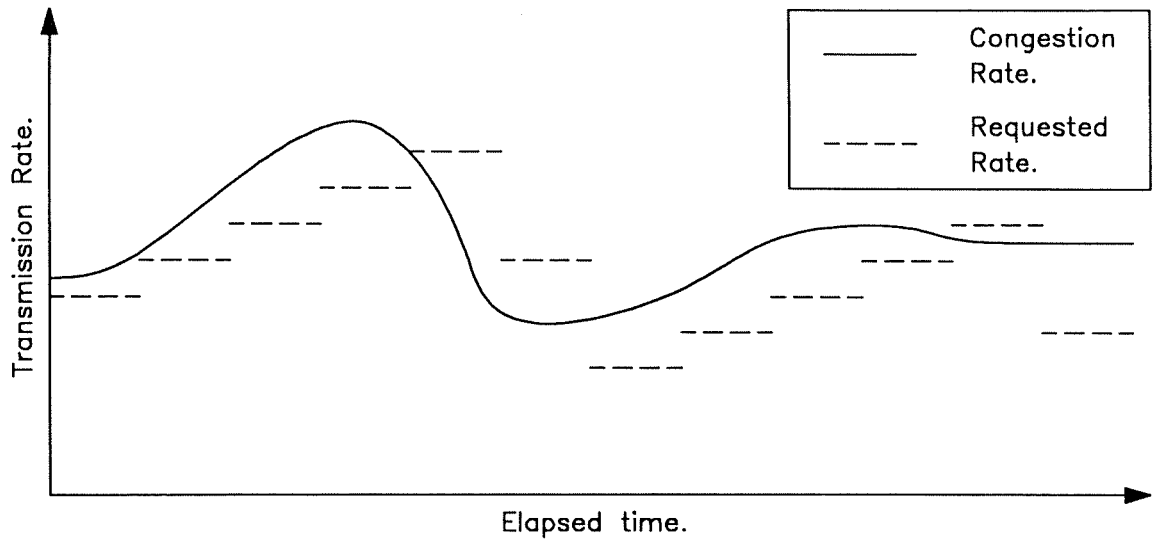
The justification of the first rule is: DATA-TPDUs are regularly lost when the transmission rate is higher than the congestion one, so the expected transmission rate has to be decreased. The reason of the second rule is: no DATA-TPDU is lost when the transmission rate is lower than the congestion one, and so increasing the expected transmission rate makes it closer to the congestion one. The application of these two rules makes the transmission rate vary just below the congestion rate, as shown on figure IV.5.

The set of all these elements, that is to say:

- using an explicit transmission rate for transmitting DATA-TPDUs,
- performing in-band retransmission of lost DATA-TPDUs,

---

15. This information can be expressed as the transmission rate itself (an amount of packets per period of time), or as the 'inter-TPDU gap' (an amount of time).



- Figure IV.5 -
The progression of the expected transmission rate.

- making the transmission rate lower than the congestion rate, but close to it,

is often referred to as the 'rate-based flow control' mechanism.

There is a major problem when implementing this transmission rate. The transport agent of the sending entity will have to wait for a little period of time between each DATA-TPDU transmission operation. It will have to do so by using a timer of the operating system. And the problem is that the required delays are often smaller than the granularity of those operating system driven timers.

To circumvent this annoying problem, most transport protocols implementing rate-based flow-control do not control the transmission rate of isolated DATA-TPDUs, but the transmission rate of short bursts of DATA-TPDUs. So, the information to control the transmission rate, which is included in each CONTROL-TPDU the transport agent of the receiving entity transmits to its peer, is composed of two values:

- the burst size, expressed as a quantity of DATA-TPDUs,
- the burst transmission rate, expressed as the quantity of bursts to be transmitted each period of time, or expressed as a delay between each burst transmission.

For example, here is a situation where a transport agent has found that:

- it can treat a thousand DATA-TPDUs every second;

- it has buffering capacity for ten DATA-TPDUs[16];
- bursts of ten DATA-TPDUs can be transmitted by the network layer, without leading to any congestion situation.

Instead of instructing its peer to transmit a thousand DATA-TPDUs every second, it can instruct the latter to transmit a hundred of bursts of ten DATA-TPDUs every second. The advantage is that the transport agent of the sending entity will be allowed to use timers with a granularity in the range of a hundredth of a second instead of a thousandth of a second.

IV.3. Examples of the improved design.

The three main examples of transport protocols using this improved solution are NETBLT, VMTP and XTP. They all come from the DOD world.

* NETBLT and the improved design.

Historically, the NETwork BLock Transfer protocol (NETBLT)[17] was the first transport protocol for which the improved design was implemented. In fact, NETBLT was the first transport protocol explicitly designed for bulk data transfer, using network routes with very long latencies and/or heavy loaded networks.

NETBLT uses the selective retransmission on demand scheme. It views the data stream to be transmitted as a sequence of blocks of DATA-TPDUs. Each block of DATA-TPDUs is known as a 'buffer' in the articles about this transport protocol. The receiving NETBLT transport agent transmits its transmission authorizations on the buffer basis: when there is buffering space for a new buffer, a special purpose CONTROL-TPDU known as 'GO[N]'[18], is transmitted to the peer NETBLT transport agent. The positive acknowledgement for all the DATA-TPDUs of a single buffer is known as a 'OK[N]', and the demand for the retransmission of some DATA-TPDUs of a buffer is known as a 'RESEND[N]'.

NETBLT uses the rate-based flow control mechanism, at the burst level. In each CONTROL-TPDU is included a 'burst size' and a 'burst rate' values.

---

16. A buffer for storing received DATA-TPDUs between the moment they are received and the moment they can be treated, not the buffer for storing data between the moment a DATA-TPDU is treated and the moment its data part is delivered to the destinating entity.
17. A description of NETBLT design may be read in [NETBLT], and the specification of this protocol may be found in [RFC-998].
18. N is the sequence number of this buffer.

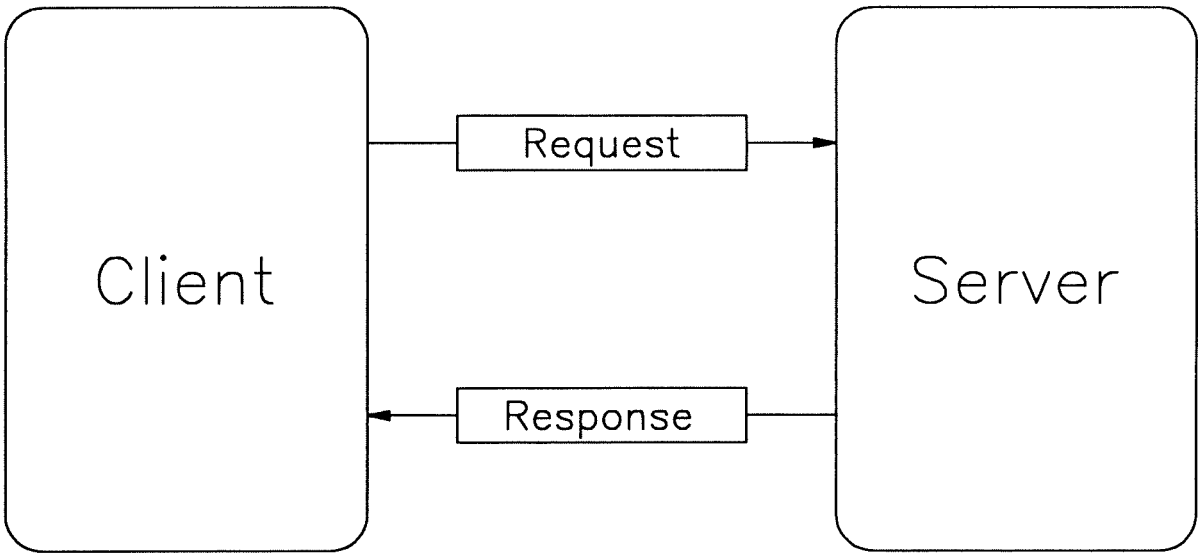
NETBLT uses a very ingenious scheme to deal with the high sensitivity to CONTROL-TPDUs loss. It could be described as a cumulative acknowledgement scheme with high redundancy in transmission. All CONTROL-TPDUs (GOs, OKs and RESENDs mixed up) have got increasing sequence numbers. Each DATA-TPDU contains a field reflecting the higher sequence number of the received CONTROL-TPDUs. And each time a new CONTROL-TPDU is to be transmitted, all non-acknowledged ones are retransmitted, all packed into a single large CONTROL-TPDU.

*** VMTP and the improved design.**

The Versatile Message Transaction Protocol (VMTP)[19] is a transaction-oriented transport protocol, especially designed for the implementation of the client/server model.

In VMTP, the communication between two entities is asymmetric, with a client entity having a sequence of transactions with a server entity. This transaction model is depicted in figure IV.6. Each transaction is composed of a 'request' message generated by the client entity and destinated to the server one, and a 'response' message transmitted back to the client entity[20]. One of the requirements of VMTP is that these messages must be transmitted with a high throughput. This

- Figure IV.6 -
The transaction model of VMTP.



19. The specification of VMTP may be read in [RFC-1045].
20. This is the basic model of a transaction in VMTP. Other models exist, such a transaction with a request but no response, or the server a group of entities instead of a single one, etc.

last point of view motivated its creators to use the improved design described in this chapter.

Each message may be up to four megabytes large, and is viewed by VMTP as a sequence of 'packets groups'. Each packets group is sixteen kilobytes long, except the last one, which may be shorter. Each such packets group may be transmitted as several DATA-TPDUs, simply called 'packets'.

The acknowledging scheme is quite a bit elaborate, with many implicit acknowledgements, cumulative acknowledgements and retransmission on demand. The simplest case occurs when no packet at all is lost: in this case there is no CONTROL-TPDU transmitted for acknowledgement purpose. Instead, the acknowledgement is implicit, in the sense that the next message[21] acknowledges the entire current one. On the other hand, when some packets of a packets group are missing, the receiving VMTP transport agent requests their retransmission, just as it has been described in the preceding paragraph. The only difference is that a request for the retransmission of some packets of a packets group also acknowledges all the preceding packets groups. And so this request is interpreted by the sending VMTP transport agent as some kind of 'selective-go-back-N': the requested packets are retransmitted, as well as all subsequent packets groups.

The justification of such an acknowledging scheme is that in the context of the transaction model, most of the messages are expected to be shorter than sixteen kilobytes. In this particular case, every message is transmitted as a single packets group, and so the acknowledging scheme virtually becomes a bare retransmission on demand one.

VMTP also uses the rate-based flow control scheme. But it uses the basic approach of that strategy: the transmission deals with the packets, and not with bursts of packets. Each request packet mentions the expected inter-packet gap for the corresponding response message, and each response packet mentions the expected inter-packet gap for the next request message.

Also, VMTP makes no effort to insure the delivery of requests for retransmission. Instead, it minimizes the traffic overhead in case of loss of such a request. When the retransmission timer of the sending VMTP transport agent expires, only one packet of the first non-acknowledged packets group is retransmitted, with a particular bit set in the Flags fields. The bit is referred to as the 'APG bit'[22]. On reception of this packet, the receiving VMTP transport agent immediately transmits back the adequate retransmission

---

21. The response corresponding to a request, or the next request following a response.
22. For Acknowledge Packet Group bit.

request[23]. On reception of this request only, the sending VMTP transport agent actually starts a possible retransmission.

*** XTP and the improved design.**

The eXpress Transfer Protocol (XTP)[24] is a brand new general-purpose transport protocol for the DOD world. The goal of its creators is to make this protocol the successor of TCP. For that reason, XTP addresses all the questions raised by the use of TCP. And so it implements many new functionalities and its design has the aim of high throughput for data transfer. It is no surprise that XTP includes the improved design.

XTP uses the rate-based flow control. It does so just as NETBLT: the receiving XTP transport agent computes an expected transmission rate for short bursts of DATA-TPDUs. And each CONTROL-TPDU it transmits to its peer includes a burst size and a burst rate values.

XTP also uses the periodic resynchronization scheme, with selective acknowledgements and retransmissions. When it has a user message ready to be transmitted, and some transmission authorization to transmit it, it starts the transmission of all DATA-TPDUs of that message. In the last DATA-TPDU of the message, a special bit is set[25]. On reception of this DATA-TPDU, the resynchronization takes place as described in this chapter.

IV.4. Results of the improved design.

The creators of NETBLT have already implemented and tested an implementation of their transport protocol. The main problem is that this implementation uses as underlying hardware:

- not extremely fast computers: PC-ATs,
- a very poor Ethernet adapter, with only a single transmission register, used for both emission and reception.

The consequence of the use of such an Ethernet adapter is that many datagrams get lost, as it becomes deaf to the network while it has a datagram ready to be transmitted.

Nevertheless, the results of NETBLT sound very good, with a measured throughput varying in the range 1.1 to 1.8 megabits per second[26]. That is quite satisfying, in comparison to the

---

23. A positive acknowledgement is then generated as a 'retransmit no packet from the packets group number N', where N is the sequence number of the last received packets group.
24. A comprehensive specification of XTP may be read in [XTP-1].
25. In the XTP articles, this bit is referred to as the <SREQ> bit, for Synchronize REQuest.
26. The interested reader may find in [PERFORMANCE] the commented results of these throughput measurements.

results of TCP, one to three megabits per second, for implementations running in far better hardware environments.

NETBLT has also been tested in a large range of difficult network conditions[27]. An example of such a difficult condition is:

- the sending and the receiving NETBLT transport agents are located on two separate LANs,
- these two LANs are connected by IP routers communicating via a satellite channel.

This environment leads to a very long RTD. The results of these tests show that NETBLT behaves very well on these extreme conditions: there is no drastic drop for the measured end-to-end throughput.

The results of VMTP are of no interest to the current chapter, as this transport protocol also uses the solution described in the next chapter: VMTP is in fact a lightweight transport protocol.

Just the same comment can be made about XTP: it is also a lightweight transport protocol. Furthermore, this protocol has been so recently defined that there is not yet any working implementation of it. And so, there is not yet any throughput measurements carried out for XTP.

So the following analysis of the improved design is only based on NETBLT results.

IV.5. Analysis of the improved design.

From its principles, the improved solution does not restrain itself in the use of the available network bandwidth. Nevertheless, the end-to-end throughput is measured far below the network bandwidth.

It has already been stated earlier in this document that the throughput of the transport layer is dependant of the most constraining resource from two limited ones:

- the bandwidth of the network connecting the communicating hosts,
- the processing capacity of the communicating hosts.

And the available network bandwidth is not a constraint.

The conclusion of all these observations is: the available computing resource of communicating hosts is the real constraint for the transport layer to achieve high throughput.

---

27. The complete description of these tests, and their results can be read in [RFC-1030].

IV.6. Conclusion.

The improved design really helps the throughput not to drastically decline in the case of very long network latency and in the case of near congested networks. On the other hand, it leads to almost no improvement in the favourable cases. The reason is: the throughput of the transport layer is in fact only limited by the computing capacity of the communicating hosts.

So, in a way to improve the throughput of the transport layer, there is a real need to find solutions that rationalize the use of the available processing resource.

Chapter V:

The lightweight transport protocols concept.

V.1. Aims of this solution.

The aim of the lightweight transport protocols concept is to provide new protocol designs, which can 'naturally' lead to implementations requiring less computing resources for each transmitted TPDU. As it has been shown in the preceding chapter, only such implementations allow high end-to-end throughputs.

The lightweight transport protocols try to meet their aim by being much simpler than the classical ones. This protocol simplification can come from:

- the simplification of the associated state machinery,
- the restriction on the number of different TPDUs types,
- the use of a common format for each TPDUs types.

An interesting analogy can be made with the RISC[1] processors technology. These processors only offer a simple instructions set to the program designers. But the creators of this kind of processors expect that this simplicity can allow very fast implementations. This situation is also true for the lightweight transport protocols: their creators believe their simplicity to vouch for very fast implementations.

V.2. First case study: VMTP.

A first case, which study is very interesting in the scope of the lightweight transport protocol concept, is the design of

1. Reduced Instruction Set Computer.

VMTP[2]. It is a very innovative one, that rejects all traditional layering principles with a massive use of recursion. And it is this recursion technique that leads to a minimum number of TPDU types, and to a simple state machinery.

VMTP uses two basic TPDU types when handling the normal condition for a transaction between a client entity and a server entity. The first TPDU type is the 'request packet', used for the transmission of a request from the VMTP transport agent of the client entity to the one of the server entity. The second TPDU type is the 'response packet', used for the transmission of a response from the VMTP transport agent of the server entity to the one of the client entity.

*** The early versions of VMTP.**

In the early versions of VMTP, six more TPDU types where used, for handling all exceptional situations. And the only existence of these numerous TPDU types was leading to a pretty complex state machinery.

A first example of such a TPDU type was the one used for requesting the selective retransmission of some TPDUs from a request or a response. In the VMTP terminology, it was known as the 'retransmission-request packet'. An error recovery is quite a natural aspect of any transport protocol. So the existence of this 'retransmission-request packet' had no real impact on the complexity of the state machinery of VMTP. Nevertheless, this TPDU type existed, and so the VMTP transport agents had to implement a function identifying and processing such particular TPDUs.

The second example of these extra TPDUs types has an annoying impact on the complexity of the protocol state machinery. These TPDUs types are linked to the addressing scheme used by VMTP. The latter is quite uncommon, due to the fact that VMTP addresses are totally independent of network (IP) addresses[3].

When using such addresses, there is a need for an address resolution scheme. It must allow any VMTP transport agent to bind any VMTP address to the IP address of the corresponding VMTP transport agent.

For this purpose, there were two special TPDUs types in the early versions of VMTP: the 'entity-information-request packet' and the 'entity-information-response packet'. When a client entity delivers a request to its VMTP transport agent, and when

---

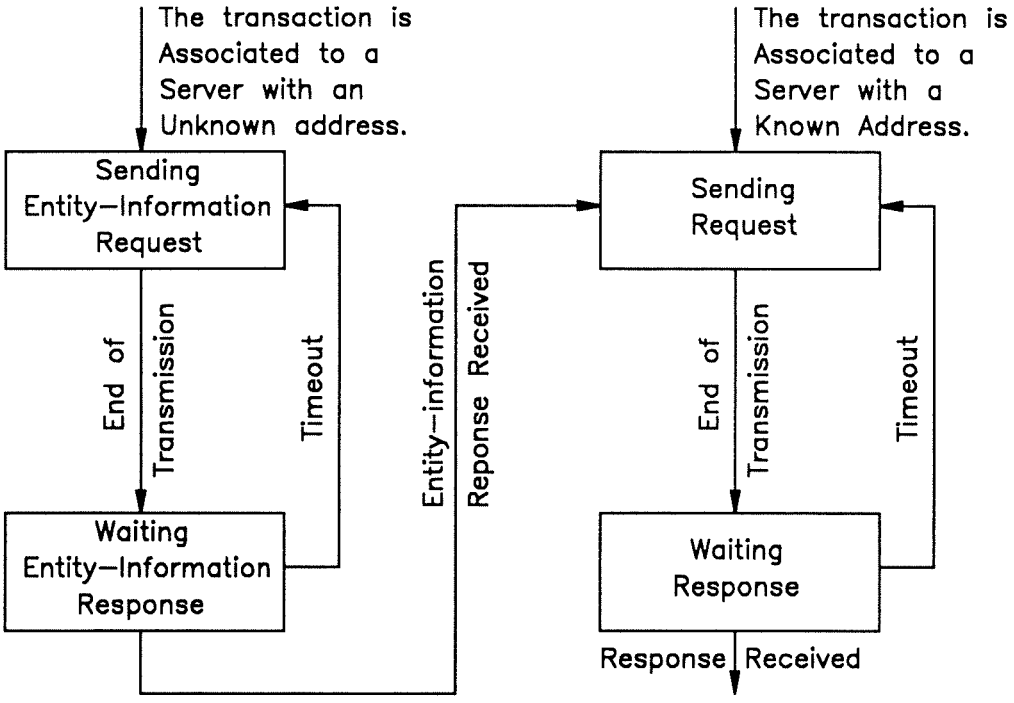
2. VMTP has been briefly described in paragraph IV.3, under the title 'VMTP and the improved design'. Its complete specification may be read in [RFC-1045].

3. In VMTP, the transport address is no more composed of a network address and a local discriminant. It is an improvement
of those fixing TCP's lack of functionality.

this request is destinated to a server entity this transport agent does not know the corresponding network address of[4], the following procedure was executed: an 'entity-information-request' was multicasted to the group of all VMTP transport agents[5], until an 'entity-information-response' was received. The originating network address of this response was in fact the searched network address, as only the VMTP transport agent of the server entity could have transmitted this response.

The use of this address resolution scheme, and so the use of these two special purpose TPDU types, leaded to a more complex state machinery for the transport agent of a client entity. This complexity is depicted by figure V.1: the first state of a transaction is 'sending request' or 'sending entity-information-request', according to the fact that the transport agent knows or does not know the IP address of the server entity. There is a first little state machinery associated to the need for a retransmission on timeout of the entity-information-request

- Figure V.1 -
The complexity of the early VMTP state machinery.



4. On the other hand, a VMTP transport agent always knows which network address to transmit a response to, as this network address comes with the associated request.
5. VMTP is built over the multicast extension of IP. This network protocol is similar to IP, except for the fact that a destinating network address can be a 'hosts group address'. In this case, the network layer tries, also on a best-effort basis, to deliver a copy of this datagram to each host which is a member of the destinating group. Multicast-IP is specified in [RFC-1030].

packet until the corresponding entity-information-response is received, and then the basic state machinery associated to the handling of the basic VMTP transaction.

*** The lightweight versions of VMTP.**

A significant simplification of VMTP comes from a single remark: each exceptional situation is solved by having some kind of a transaction with the VMTP transport agent of the peer entity, or with the group of all VMTP transport agents.

So it is very helpful to consider that there is a special VMTP entity associated to each VMTP transport agent. This entity, in the VMTP terminology, is referred to as the 'VMTP manager'. Its characteristic is that it is the only entity which have an access to the data structures of its local VMTP transport agent.

The transport service of VMTP is then provided in the following fashion. The VMTP transport agents insure the handling of normal transactions, only using the 'request packet' and the 'response packet'. They handle all unusual situations by simply calling their local VMTP manager. The VMTP managers perform the handling of these exceptions, by:

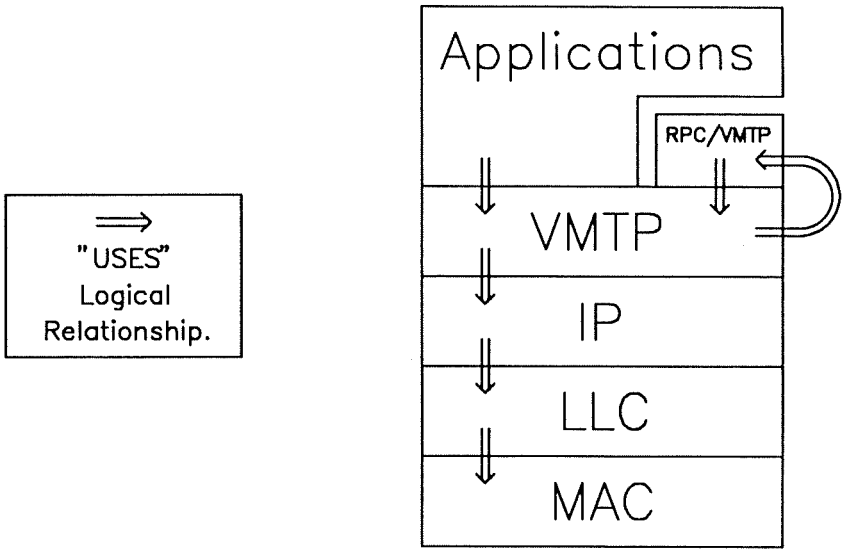
- having transactions ones with the others,
- by manipulating the data structures of their local VMTP transport agent.

This situation is quite strange, as it totally violates all the basic layering concepts, as shown on figure V.2. The transport layer provides its service:

- by having its transport agents communicating using a transport layer protocol,
- by using the network layer service,
- by using the transport layer service through a customized RPC presentation (this is the recurrent aspect of VMTP).

Furthermore, a VMTP manager is a user of this transport layer; but it does not consider the latter as a black-box: it even manipulates its internal data structures.

The substitution of the former 'retransmission-request packet' is performed as follows. When a VMTP transport agent finds its the proper time to ask for the retransmission of some packets of this message, it simply calls its VMTP manager. The latter then has a transaction with the VMTP manager local to the sending VMTP transport agent. This transaction simply asks this VMTP manager to make its local VMTP transport agent retransmit



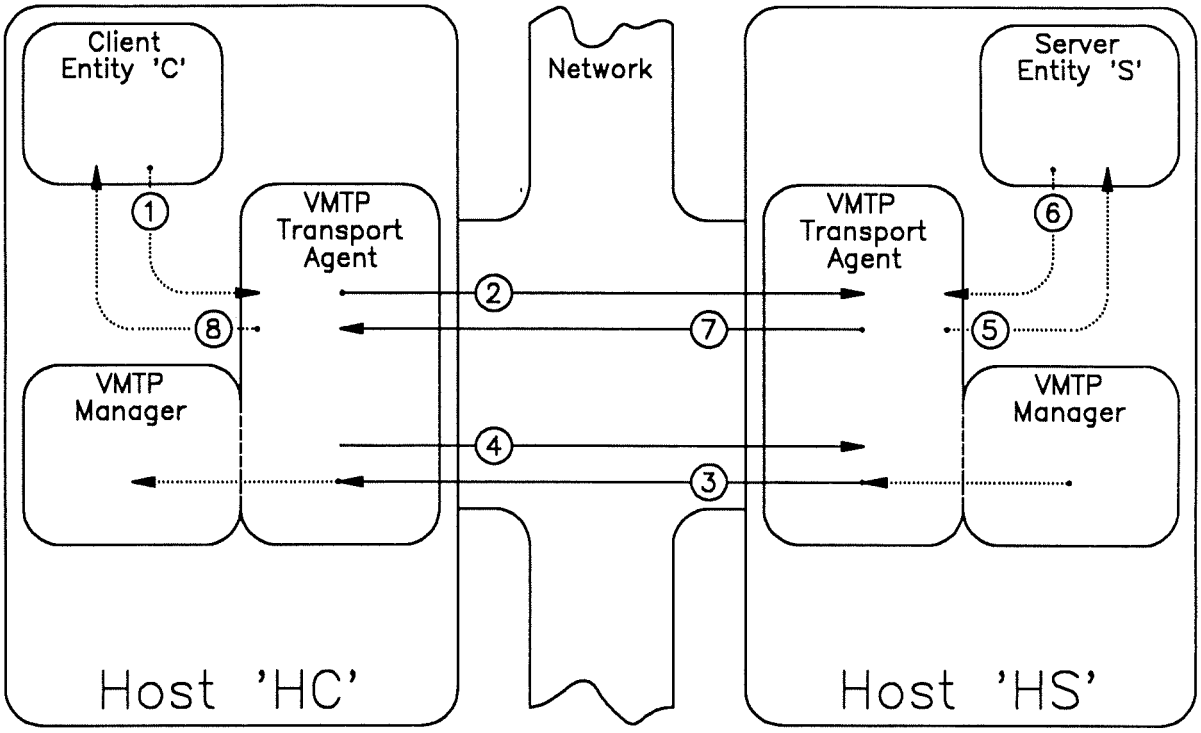
- Figure V.2 -
The use of recursion by the VMTP protocol.

the missing packets[6]. It does so by updating the state information for that message, and by sending a signal to the VMTP transport agent. The retransmission is then performed.

This is represented in figure V.3:
(1) The client entity 'C', running on host 'HC', requests its VMTP transport agent to have a transaction with the server entity 'S', and delivers the appropriate request message.
(2) The VMTP transport agent of host 'HC' transmits this request message to the VMTP transport agent of host 'HS', where the entity 'S' runs.
(3) The VMTP manager of host 'HS' requests the VMTP manager of host 'HC' to make its local VMTP transport agent retransmits the missing parts of the last request message from entity 'C' to entity 'S'.
(4) The VMTP transport agent of host 'HC' retransmits these missing parts of the request message to the VMTP transport agent of host 'HS'.
(5) The VMTP transport agent of host 'HS' delivers the request message to entity 'S'.
(6) The entity 'S' delivers the corresponding response message to its local VMTP transport agent.
(7) The VMTP transport agent of host 'HS' transmits this response message to the VMTP transport agent of host 'HC'.
(8) The VMTP transport agent of host 'HC' delivers the response message to entity 'C'.

---

6. This transaction uses one of the optional models of VMTP transactions: the datagram one, where a request is transmitted once and then never retransmitted, and where there is no expected response.

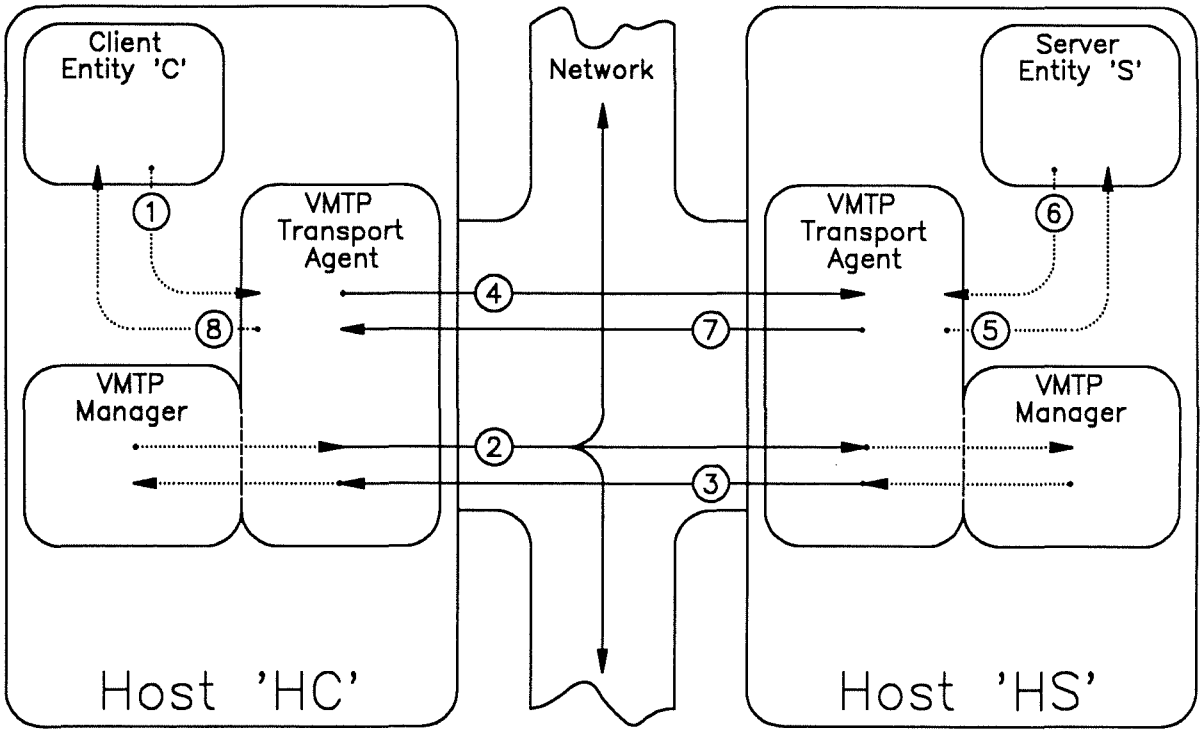


- Figure V.3 -
The retransmission requests in VMTP.

The substitution of the former address resolution protocol occurs as follows. When a VMTP transport agent has to transmit a request to a server entity it does not know the IP address of, it simply asks it to its local VMTP manager. To provide this information, the latter has a transaction[7] with the entire group of VMTP managers, requesting all the information concerning the searched server entity. The unique response comes from the VMTP manager local to the VMTP transport agent servicing this entity. And the originating IP address of this response is in fact the IP address of the above server entity.

This is represented in figure V.4:
(1) The client entity 'C', running on host 'HC', requests its VMTP transport agent to have a transaction with the server entity 'S', and delivers the appropriate request message.
(2) The VMTP manager of host 'HC' requests the group of all VMTP managers for the informations about entity 'S'. This is done using a VMTP request message.
(3) The VMTP manager of host 'HS' responds these informations, using a VMTP response message.

7. This transaction uses one of the optional models of VMTP transactions: the multicast with only one response expected one, where the request is multicasted to a group of server entities and then retransmitted on timeout, until a response comes from one of these server entities.



- Figure V.4 -
The address resolution scheme of VMTP.

(4) The VMTP transport agent of host 'HC' transmits this request message to the VMTP transport agent of host 'HS' (as host 'HS' runs 'S').
(5) The VMTP transport agent of host 'HS' delivers the request message to entity 'S'.
(6) The entity 'S' delivers the corresponding response message to its local VMTP transport agent.
(7) The VMTP transport agent of host 'HS' transmits this response message to the VMTP transport agent of host 'HC'.
(8) The VMTP transport agent of host 'HC' delivers the response message to entity 'C'.

From the description of these two examples, the essential characteristics of the lightweight design of VMTP can be pointed out. The VMTP managers exchange messages with exactly the same semantic as the one of the messages the VMTP transport agents exchanged in the earlier versions of the protocol. But the underlying objects to transport these messages have moved from special purpose packets to the basic request and response ones. In fact, in the new design of VMTP, all protocol handling is realized with these two basic TPDU types. Furthermore, these two TPDU types have exactly the same format, with the same fields, which have the same semantic. They can only be distinguished thanks to the value of a single bit in the header,

indicating the packet type. So it can be considered that there is a single TPDU type, with two variants:

- the first variant, which is transmitted from the VMTP transport agent of a client entity to the one of a server entity, is the request packet,
- the second variant, which is transmitted in the opposite direction, is the response packet.

A consequence of this reduction of the quantity of TPDU types is a radical simplification of the VMTP state machinery. As a matter of fact, some special purpose TPDU types incurred some new states and state transitions[8]. In the lightweight version of VMTP, these special purpose TPDU types have been replaced by embedded transactions. And so their associated parts in the VMTP state machinery have vanished: they are in fact handled by other occurrences of this simplified VMTP state machinery.

V.3. A second case study: XTP.

An other interesting transport protocol, in the scope of the lightweight transport protocols study, is the case of XTP[9]. The service this protocol provides is far more traditional than the one VMTP offers: it transports user messages on a full-duplex connection established between two entities. Even if this situation does not lead to quite obvious simplifications, the creators of XTP claim this transport protocol is a lightweight one.

First of all, all XTP-PDUs types have the same layout composed of three parts:

- the common header, which is twenty-four bytes long,
- the segment, which is a multiple of height bytes long,
- the common trailer, which is sixteen bytes long.

As they names indicate, the common header and the common trailer have exactly the same format for all XTP-PDUs types. A single field in the common header, called the CMD field, has got a value identifying the type of a XTP-PDU. Only the content of the remaining part of this XTP-PDU is function of its type: the segment.

Second, an important aspect of XTP is the length of all parts of a XTP-PDU. They are all a multiple of height bytes. This is very interesting for the implementation purpose. All three XTP-PDU parts are aligned at height bytes boundaries[10].

---

8. An example, depicted by figure V.1, has been described earlier in this paragraph.
9. XTP was briefly described in paragraph IV.3, under the title "XTP and the new design". An overview of this protocol can be read in [XTP-2], and its complete specification can be found in [XTP-3].
10. So the segment part may sometimes be ended with some padding bytes to reach the next eight bytes boundary.

So they are handled by efficient softwares, even when the host uses a processor with general-purpose registers up to sixty-four bits.

A counter-example will illustrate the importance of this aspect of XTP. If the parts of the XTP-PDU were aligned on two bytes boundaries, for example, there would be some problems with the implementation destinated to some processors. As a matter of fact, some thirty-two bits processors[11] can only read or write:

- four bytes long words which are aligned on four bytes boundaries,
- two bytes long words which are aligned on two bytes boundaries,
- one byte long words which are aligned on one byte boundaries.

For processing the TPDUs with this processor, only sixteen bits long registers can be used, as the parts of the TPDUs are aligned on two bytes boundaries. Such an implementation misses the chance of using the thirty-two bits long registers, which would allow faster computing, as they would incur about half as much interactions with the memory.

V.4. Results of the solution.

* The results of VMTP.

Some throughput measurements have been carried out for VMTP. Their results are very interesting for the analysis that will be done in the next paragraph.

The tested implementation of VMTP uses a ten megabits per second Ethernet LAN, and may be executed on a large range of computers, from workstations to mainframes. The results of the throughput measurements roughly vary between two and four and a half megabits per second[12].

* The results of XTP.

There are no available throughput measurements for the protocol XTP so far. The reason of that fact is quite simple: at the current time, no implementation of this protocol is operative.

---

11. Some RISC (Reduced Instruction Set Computer) processors for instance).
12. The complete results of these measurements can be read in [PERFORMANCE].

V.5. Analysis of the lightweight protocols concept.

The results that have been announced in the preceding paragraph lead to some comments.

There is an undisputable throughput improvement, from TCP to VMTP. In general, TCP implementations allow throughputs ranging from one to three megabits per second, and VMTP ones allow throughputs ranging from two to four and a half megabits per second. Furthermore, it must be pointed out that VMTP implementations are still very young in comparison to those of TCP[13]. So the throughput breakthrough between these transport protocols is likely to become more and more radical.

On the other hand, it must also be noticed that, at this point, VMTP uses two solutions that TCP does not:

- VMTP uses the improved design,
- VMTP is a lightweight transport protocol.

And, in the throughput improvement from TCP to VMTP, it is very difficult, if not impossible, to evaluate the benefit introduced by the sole lightweight protocol concept.

But, eventually, a concluding observation is that even for a lightweight transport protocol, all the available network bandwidth is not yet used to transfer user messages. So, in other words, such protocols also lead to implementations consuming too much computing resources.

V.6. Conclusion.

The lightweight transport protocol concept meets its aims, as it helps reducing the processing overhead. Nevertheless, this reduction is really not as important as it could be expected: the end-to-end throughput remains far beyond the underlying network bandwidth. The only explanation of this half failure is: the computation overhead does not come from the protocol processing itself, but from the processing of functions which are peripheral to the protocol processing.

So, to design a really fast implementation of a transport agent, there is a need to focus with more care on these peripheral functions than in the transport protocol definition.

---

13. TCP is now about fifteen years old, but the latest versions of VMTP specifications are quite recent.

Chapter VI:

Optimized implementations.

VI.1. Aims of the solution.

The aim of the optimized implementations is to reach higher throughputs with a transport protocol without changing its design. As the key constraint is the available processing time for handling exchanged TPDUs, the solution is to focus on the implementation of the transport agents, for the purpose of optimizing it.

Two general tendencies exist in optimization: the macro-optimization and the micro-optimization.

The aim of micro-optimization is to look for the best implementation of an algorithm, once the algorithm has been chosen. The procedure to realize it is to search for the best data structure to handle, and the best program to compute the algorithm. The most important aspect of this process is that every detail must be reviewed, even the deepest one.

A characteristic of micro-optimization is that it must take into account all the interesting distinctive features of the underlying computing system. Because of this property of 'subordination to the actual machine', the micro-optimization aspect will not be discussed in this document.

The aim of macro-optimization is to look for the most appropriate algorithm to realize a given task. At this level, the decisions are not based on 'machine-dependant' issues, but often on conclusions read in the theory of the algorithmic complexity. This is the tendency of implementation optimization that is discussed in this chapter.

VI.2. Some optimization principles.

Here comes an analysis of some of the most representative macro-optimization principles. The aim is not to be exhaustive, but to show that there is a great difference between a traditional implementation of the specification of a transport protocol and an optimized one.

In fact, only three such optimization principles are analyzed here. They are the 'template header caching', the 'fast TCB look-up' and the 'packet prediction'.

* The template header caching principle.

This first analyzed macro-optimization principle concerns the emission of DATA-TPDUs, and has as aim to avoid redundant computing. Furthermore, it does not only involve the implementation of the transport agent, but the one of the network agent as well, or to be more exact it influences the cooperation between these two pieces of software[1].

The general algorithm used to emit a part of a user message into a DATA-TPDU is the following one:

- prepare a DATA-TPDU, by filling in all its fields with the appropriate values,
- deliver this DATA-TPDU to the network layer.

A very dumb implementation of the transport agent would use this algorithm right to the letter. It would build the new DATA-TPDU in some free memory space, by computing in sequence the value of each of its fields and storing the results at the appropriate memory locations. It would then deliver this newly built DATA-TPDU to its network agent, mentioning the memory location of that DATA-TPDU as well as the network address of the destinating transport agent.

For a slightly better implementation, one could notice that a non-negligible part of the TPDU header is common to all emitted DATA-TPDUs. For example, the Source Port and Destination Port fields[2] are the same for all DATA-TPDUs of a given connection[3]. The value of the Flags field is also common to almost all DATA-TPDUs: it only changes in some of them to reflect some exceptional conditions such as 'this is the end of the message' or 'an acknowledgement is requested'. In most transport protocols there is also a field mentioning the version number of the protocol, and another one indicating the TPDU

---

1. A comprehensive description of this principle may be read in [TCP-1], in its eighth section, named "Output processing: an implementation trick".
2. They are sometimes transport addresses instead of port numbers in newer transport protocols such as VMTP. This has been discussed in the preceding chapter.
3. For the remaining part of this document, the word 'Connection' will be used in its general sense. It can be an explicitly established connection in the case of connection-oriented transport protocols, or an implicit connection for two entities regularly exchanging messages using a connection-less transport protocol.

type. The value of these fields are obviously constant for all emitted DATA-TPDUs. In fact, only two fields vary from DATA-TPDU to DATA-TPDU: the sequence number and the checksum ones, and all the other ones are either constant or only vary in some exceptional conditions.

So a rational implementation of the specification of the transport protocol must take this aspect into account. For each connection, a template header could be computed and then stored in the data structures of the transport agent. In this template, all the constant fields could be initialized with their appropriate value, and all the fields that only vary from time to time could be initialized with their default value. The DATA-TPDU preparation step would then be replaced by the following ones:

- Block-copy the template header into the new DATA-TPDU.
- Complete the new DATA-TPDU by filling the non-initialized fields, and by overriding those which value must be different from the default one.

It is rather clear that the second algorithm is faster than the first one. Initializing a field with a constant value can be viewed as moving that value from one memory location to the other. So initializing in sequence all fields with a constant value can be viewed as moving in sequence several parts of the memory. But all these parts have very little lengths: eight or sixteen bits in most cases, very sporadically thirty-two bits. If the processor of the machine uses general-purpose registers of sixteen, thirty-two or even sixty-four bits, a single copy instruction of the block-copy operation can thus move several field values at once: this is faster than separately moving them.

This optimization only becomes really interesting if its underlying reasoning is also applied more deeply, namely: to the network layer. As a matter of fact, the work of the network layer is:

- to prepare a datagram, which consists of adding a header to the TPDU,
- to take a routing decision, which consists of choosing one of the local link layer agents and a destinating address on that link,
- to deliver the destinating address and the datagram to the chosen link layer agent.

For a given connection between two entities, the minimal aspect of the network layer ensures that most of the fields of the added header have the same values for all transmitted TPDUs. Some of these fields only vary from time to time, according to the routing decision, for example. These routing decisions themselves are also due to give the same results for long sequences of TPDUs. So using the traditional model of the transport agent delivering a sequence of TPDUs to the network

agent, each one independent from the others, leads to some processing overhead. It would be interesting to cache, for each connection the transport agent handles, the information the network agent computes, for the purpose of not recomputing it for each transmitted TPDU of that connection. But the problem is that the minimal network layer does not know this concept of connection.

The solution to circumvent this problem is to redefine the physical interface between the transport agent and the network agent. The logical interface between these two modules is composed of the N-DATA.REQUEST and N-DATA.CONFIRMATION services[4]. In traditional implementations of the network agent, both services are mapped to a single function, say NetSend. Here is what the specification of this function looks like:

Function header:
- NetSend(NAdr,Data,Status).

Input parameters:
- NAdr, a network address,
- Data, an array of bytes.

Output parameters:
- Status, an integer.

Specification:
- NetSend tries to send the content of Data to the host with the network address NAdr. If it manages to send the appropriate datagram, Status is set to TRANSMITTED; in the opposite case, Status is set to FAILED.

A more interesting implementation of the network agent would offer two separate functions, say NetTemplate and NetFastSend. Here is what their specifications could look like:

Function header:
- NetTemplate(NAdr,Template).

Input parameters:
- NAdr, a network address,

Output parameters:
- Template, a compound data structure.

Specification:
- NetTemplate saves in Template the template network header and the routing decision for all datagrams that would have to be transmitted to the host with the network address NAdr, as well as a timestamp.

Function header:
- NetFastSend(NAdr,Template,Data,Status).

Input parameters:
- NAdr, a network address,
- Template, a compound data structure,
- Data, an array of bytes.

4. This has been discussed in paragraph I.1.

Output parameters:
- Template, a compound data structure,
- Status, an integer.

Specification:
- NetFastSend first verifies if the information contained in Template is still valid[5]. If it is not the case, Template is recomputed. NetFastSend then tries to send the content of Data into a datagram built using the template header saved in Template, and using the routing decision also saved in Template. If it manages to send the appropriate datagram, Status is set to TRANSMITTED; in the opposite case, Status is set to FAILED.

From these specifications, it is clear that a call to the function NetSend, or a call to the function NetTemplate followed by a call to the function NetFastSend have just the same result. And it is also clear that the algorithm:

```
NetSend(hd,tpdu1);
NetSend(hd,tpdu2);
NetSend(hd,tpdu3);
```

and the algorithm:

```
NetTemplate(hd,template);
NetFastSend(template,tpdu1);
NetFastSend(template,tpdu2);
NetFastSend(template,tpdu3);
```

are also equivalent for their results, but with the second one faster than the first. So, a very good optimization of the transport agent is to implement it over a network agent providing functions equivalent to NetTemplate and NetFastSend. For each transport connection, the appropriate template could be stored in the data structures of the transport agent, and this template used for each TPDU transmission.

But there is a problem with the solution of storing the template transport header and the network template for each connection a transport agent handles. This can consume a large amount of memory. So it would be more judicious to cache all this information only for the most active connections. A good cache management policy is the classical one, where only the information for the few most recently used connections is stored.

This scheme of 'template header caching' has a very interesting property. It drastically reduces the consumption of the computing resource for the TPDUs emission. But in fact, it

5. This can be done by comparing the timestamp of Template to the one associated to the last modification of the routing informations.

is generally admitted that the main source of computing overhead is the TPDUs reception. This aspect is the concern of the two other schemes presented in this paragraph.

*** The 'fast TCB look-up' principle.**

For each active connection it handles, a transport agent must memorize some amount of information depicting the current state of this connection. This information relative to a connection is sometimes referred to as the 'transmission control block', or TCB[6].

When a transport agent is delivered a new TPDU which successfully passes the checksum verification, it has to process this TPDU in regard to the current state of the corresponding connection. To do so, the transport agent first needs to locate the correct TCB in its data structures. It is clear that this TCB location has to be very fast. The 'fast TCB look-up' principle deals with the general organization of the TCBs table, with the purpose of meeting the preceding requirement[7].

Logically, each entry in the TCB table is identified by the active connection it describes; and a connection is identified by two communicating entities, a local one and a remote one. Thus in practice, each entry in the TCB table is identified by the transport addresses of this pair of entities[8]. The TCB look-up consists, for a transport agent which has just been delivered a TPDU, to extract from the latter the transport addresses of the involved entities, and to retrieve the corresponding TCB in the TCBs table.

It is manifest that the general organization of this TCBs table has an important effect on the amount of computing resources the location of a TCB requires. As this TCB look-up is a mandatory operation for each received TPDU, reducing its computing resource consumption also helps restricting the computing time associated to each received TPDU, and thus it helps increasing the rate at which a transport agent is able to treat TPDUs.

The worst organization for a TCBs table is in fact no organization at all: a raw array of TCBs. In this case, the TCB look-up is sequential: the searched identifier is compared to the one of the first entry in the table, then to the one of the following entry in the table, and so on until a match is found. On average, half the entries of the TCBs table must be accessed

---

6. At least in the TCP terminology.
7. A comprehensive description of this principle may be read in [TCP-1], in its tenth section, named "Checksums and TCBs: the missing steps".
8. When a transport address is composed of a network address and a port number, the TCB identifier is reduced to the network address of the remote entity, its port number, and the port number of the local entity.

for each TCB look-up. This is a disaster for transport agents handling a large amount of connections.

The first improvement comes from the phenomenon of locality, which can be explained by this fact: the activity of a connection is far from being constant. This activity is in fact concentrated on periods following the transmission of a window authorization. And as a good transmission window policy requires the generation of a window authorization only when a substantial amount of buffering space is free at the transport agent of the receiving entity end[9], these periods of activity are few but intensive. For short, a connection is characterized by successions of periods with very low activity followed by short periods of very high activity. And for a transport agent handling a reasonably low number of connections, the periods of high activity of all connections come in sequence.

So there is a great chance that relatively long bursts of incoming TPDUs concern a single connection. Thus it is very interesting to have a TCBs table with the most recently used TCB sorted first. The look-up is once again sequential, but with the property that the first entry in the table is often the searched one. On the other hand, when the first TCB of the TCBs table does not match, half the table must be accessed, on average.

This solution is only valuable for transport agents handling a reasonably low number of connections. The higher the number of handled connections, the higher the probability that periods of high activity from different connections collide. In this case, the preceding solution becomes as slow as the unsophisticated sequential look-up. So there is a need to have a solution which allows a fast look-up for several amongst the most active connections.

The ideal solution for a fast TCB look-up is then to organize the TCBs table as a hashed table. This solution consists of splitting the content of the TCBs table in a set of subtables. Each TCB belongs to one of the subtables, according to the result of a 'hashing' function applied to its identifier. And each TCBs subtable is organized with its most recently used TCB sorted first. The TCBs look-up consists then to apply the same hashing function to the identifier of the searched TCB. The result of the previous computation identifies the subtable the searched TCB belongs to. This TCBs subtable is then sequentially looked-up.

This solution is very fast for many reasons. First, if the hashing function is well chosen, there is a great chance that all the very active connections have their TCBs in separate subtables. In this case, the price to locate a TCB is the one paid for:

---

9. J. Postel describes such a good transmission window policy for TCP, in [RFC-793].

- the application of the hashing function,
- the access to the first TCB of the corresponding TCBs subtable.

Second, if the searched TCB is not one of the most recently used, the price to locate a TCB is the one paid for:

- the application of the hashing function,
- the access to half the entries of the corresponding TCBs subtable, on average.

The conclusion is: a TCBs table organized as a hashed table, using a well chosen hashing function and composed of a reasonable number of subtables, is suited for a very fast look-up. This is particularly true for transport agents handling a very large number of connections.

*** The 'header prediction' principle.**

The most interesting and promising improvement for the implementation of a transport agent is also connected to the TPDUs reception, and is often referred to as the 'header prediction' principle[10].

In a traditional implementation of the transport agent, a received TPDU is processed independently from the other ones, using the algorithm which is represented on figure VI.1, and which is composed of four main steps:

- a verification of the checksum of the TPDU,
- a look-up for the TCB associated to the TPDU,
- a validation of the header of the TPDU, in regard to the content of the corresponding TCB[11],
- the real processing of the TPDU, in regard to the content of the corresponding TCB.

This last step may be more refined, by isolating the two fundamental logical steps it contains:

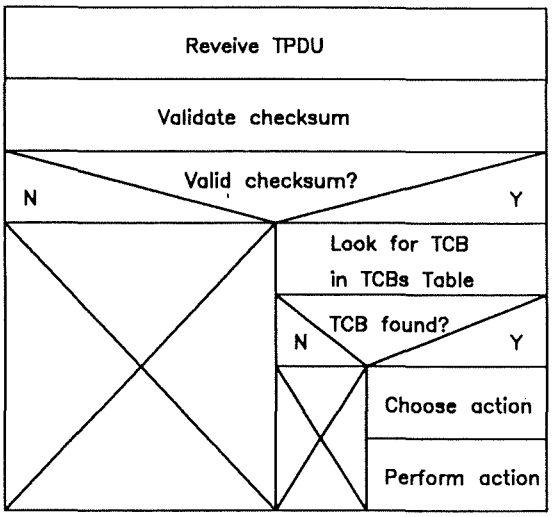
- the choice of the action to undertake with the TPDU, that is to say, the choice of a subroutine,
- the real undertaking of the chosen action, that is to say, the call to the chosen subroutine.

In physical implementation of the above algorithm, all steps are not so clearly separated. For example, the validation of the TPDU header and the choice of the action to undertake with the TPDU are both mixed up in a unique big 'decision tree', and the undertaking of the chosen action is not always implemented as a subroutine call. Nevertheless, this expression of the algorithm used to process received TPDUs is suitable for the following discussion.

---

10. A talk on header prediction has been given by Dr. Kanakia in [SLIDES], referring to unpublished work of Dr. Jacobsen.

11. It is generally admitted that a transport protocol implementation must be 'conservative'. This means that a transport agent is supposed to only transmit valid TPDUs, but is forbidden to suppose that the TPDUs it is delivered are all valid ones.



- Figure VI.1 -
GNS of the traditional algorithm
for the processing of received TPDUs.

The problem with this kind of implementation is that it misses some opportunities of anticipation. The definition of a protocol leads to some 'highly probable' chainings of TPDUs. So, when a transport agent sends or receives a TPDU, it can sometimes predict what the next TPDU to be received for that same connection will look like, at least for its header part.

Here is a non exhaustive list of such possible predictions:

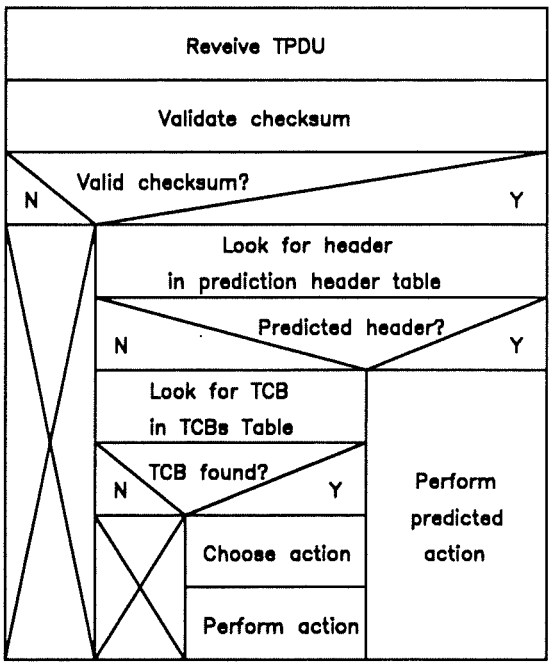
- When a transport agent has received a DATA-TPDU which is not the last one of the transmission window, it can assume that the next TPDU to be received will be the DATA-TPDU with the next sequence number.
- When a transport agent has transmitted a CONTROL-TPDU with a new transmission authorization, it can assume that it will soon receive the first DATA-TPDU of the new transmission window.
- In the Go-Back-N scheme, when a transport agent has transmitted the last DATA-TPDU of its transmission window, it can assume that the next TPDU to be received will be the CONTROL-TPDU acknowledging all transmitted DATA-TPDUs.
- In the retransmission on demand scheme, when a transport agent has transmitted the last DATA-TPDU of a block, it can assume that it will soon receive a positive acknowledgement for that block.
- In the retransmission on demand scheme again, when a transport agent has transmitted a request for retransmission, it can assume that the next TPDU to be received is the first requested DATA-TPDU.

- In the periodic resynchronization scheme, when a transport agent has transmitted the request for resynchronization, it can assume that it will soon receive a positive acknowledgement for all DATA-TPDUs that have been transmitted since the preceding resynchronization point.

A more clever implementation of the transport agent can take all these predictions opportunities into account. Whenever it can assume that it will soon be delivered a given TPDU, a transport agent can actually compute what will be the header of that TPDU. It can also easily compute the location of the TCB associated to that predicted TPDU, as it is currently using it. Furthermore, the prediction itself also includes the action to undertake with the predicted TPDU. These three informations, that is to say the predicted header, its associated TCB and its associated action, can be stored in the data structures of the transport agent.

The new algorithm for the processing of received TPDUs is shown on figure VI.2, where a new step is inserted between the checksum validation and the TCBs table look-up. The latter consists of scanning all TPDUs predictions. If the header of the received TPDU matches one of the predictions, the

- Figure VI.2 -
GNS of the predicted algorithm
for the treatment of received TPDUs.



corresponding saved action is undertaken, using the saved TCB location. On the other hand, if the header of the received TPDU does not match any prediction, the traditional algorithm is used.

This header prediction algorithm is very valuable at the computing resource consumption point of view. This comes from the following fact. When a received TPDU was predicted, the TCBs table look-up, the header validation and the action choice steps are no more executed, at the cost of a look-up in the header prediction table. And this look-up cost may be kept minimal by choosing a proper organization for this header prediction table. A hashed table, similar to the one of the TCBs table, is such a proper organization. By keeping this table reasonably short, the look-up is very fast.

The problem with the header prediction principle is to purge all out-of-date predictions from the header prediction table. This purge operations are necessary, as a prediction can sometimes be erroneous and thus never be used, and as the header prediction table must be kept short. A good solution is to cache the predictions in a header prediction table with a fixed number of entries. When a new prediction has to be inserted in the table, a free entry is used, and when there is no more free entry in the table, the new prediction replaces the oldest one. So the erroneous predictions eventually get erased from the header prediction table, which only contains the freshest ones.

Some experimentations of this header prediction principle have been carried out. They show that memorising one or two predictions for each connection leads to a quite good matching ratio[12].

VI.3. Examples of the improved implementation.

About all important transport protocols have now a version of their implementation which can be considered as an improved one. But these versions are still of experimental interest: none of them is yet commercially distributed.

As this idea of improved implementation is in fact in competition with the lightweight design tendency, it is no surprise that it started with the improvement of the classical transport protocols implementations. The main one is the TCP implementation of Dr. Van Jacobsen. It must also be noticed that efforts were also made in the case of TP-4.

---

12. This has been stated in [SLIDES], by Dr. Kanakia.

But, as the results were concluding for these classical solutions, these principles have also been extended to some implementations of VMTP, which also uses the lightweight transport protocol concept[13].

VI.4. Results of the improved implementations.

In the case of TP-4, an improved implementation has been built and tested. The results show a non-negligible breakthrough in the measured throughput:

- the normal implementations allowed end-to-end transmission rates in the range of seven hundred kilobits per second to one megabit per second,
- the improved implementation allows end-to-end transmission rates in the range of one point eight to two point eight megabits per second.

But the most interesting result comes from the optimized TCP implementation of Dr. Van Jacobsen. It runs on Sun 3/60 workstations connected by a ten megabit per second Ethernet LAN. The traditional implementation of TCP which is distributed with this machine is already considered as a very performing one, and allows an end-to-end throughput of three megabits per second. But the one of Dr. Jacobsen allows an end-to-end transfer rate of eight megabits per second, just in the same conditions.

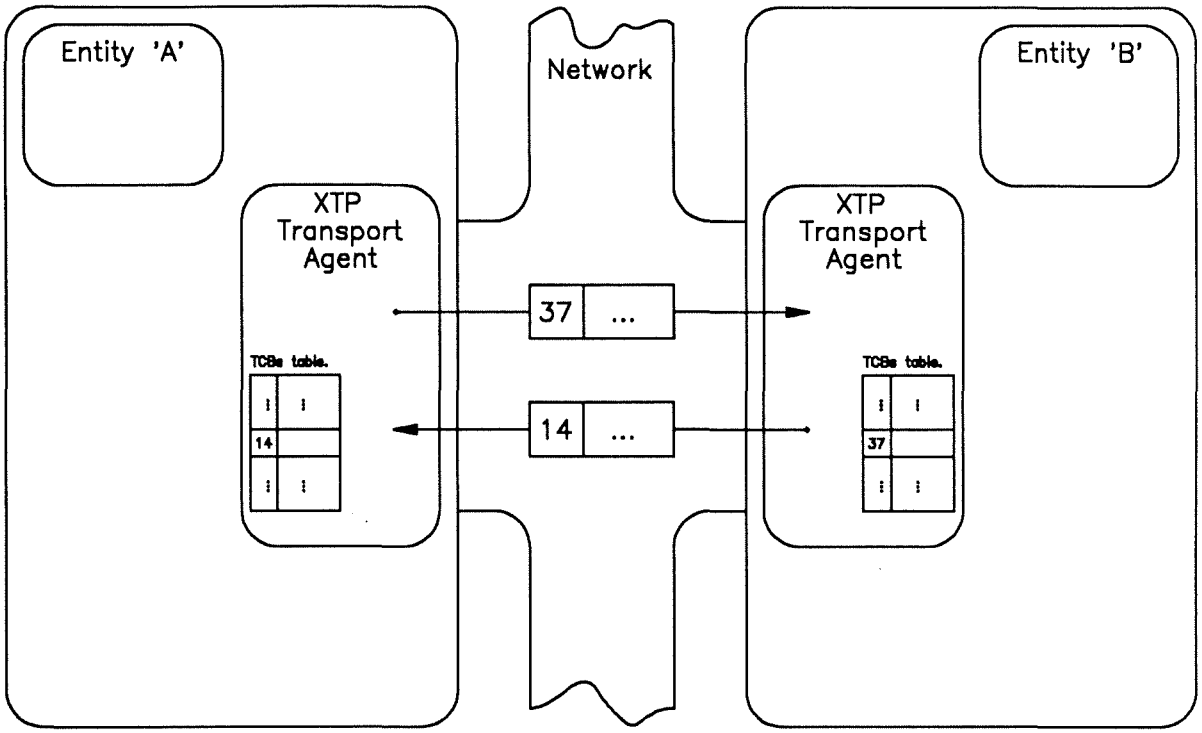
VI.5. Analysis of the improved implementation principle.

The comparison of the results of the lightweight transport protocols with those of these improved implementations lead to the following observation: most improvements of the implementations of the transport agents that lead to a real throughput enhancement concern functions that can not be considered as part of the actual protocol computing.

But this consideration does not mean that the lightweight transport protocol is totally useless. It is only insufficient. An illustration of that fact is the TCB look-up in the case of XTP. The association of the XTP-PDUs to their corresponding connection is not realized using any Source and Destination port fields, but using a unique field called the Key field. The only semantic of this field is: it must allow to identify which connection the XTP-PDU belongs to. And this identifying value could be the location of the corresponding TCB. Thus in XTP, there is no need for a fast TCB look-up, and there is no such a TCB look-up. This situation is depicted in figure VI.3.

---

13. This has been discussed in the preceding chapter.



- Figure VI.3 -
The Key field of XTP.

Entities A and B have a connection. The transport agent of A has stored the TCB of this connection in the entry number 14 of its TCBs table. The transport agent of B uses the entry number 37 for the same purpose. So all XTP-PDUs the transport agent of A transmits to the transport agent of B have their key field set to 37, and all XTP-PDUs the transport agent of B transmits to the transport agent of A have their key field set to 14.

But the interest of the improved implementations is that they show the limited responsibility of the protocol itself on the processing overhead. There are peripheral protocol-independent functions that must also be optimized.

VI.6. Conclusion.

The improved implementations principle shows a real improvement in the end-to-end throughput. But it must also be noted that even these implementations do not use all the available network bandwidth.

The conclusion is: they are a first step towards really fast transport protocol implementations. The computing overhead must be more closely analyzed, in a way to find its real causes. New implementations must then be designed, with the aim to eliminate all the discovered sources of computing overhead.

Chapter VII:

The smart network adapters.

VII.1. Aims of the solution.

The aims of the solution described in this chapter is to find implementation principles, for the transport agents of a transport protocol, which allow real high throughput. For this purpose, there will be no limitation at all on the nature of the proposed solutions: even hardware support will be studied.

The two preceding chapters highlighted an important aspect of the ideal solution: it has to optimize the computing of the peripheral functions or to avoid the use of such functions when they can not be optimized. An audit of the computing overhead all these functions generate must be taken into account, in a way to choose the most important aspects to focus on.

VII.2. Principles of the solution.

* The sources of computing overhead.

Figure VII.1 reports the audit of the TPDU processing time in the cases of two transport protocols: TCP and VMTP. These results lead to some comments that highlight the real sources of computing overhead.

The first observation is: the processing time closely associated to the protocol driving accounts for only sixteen percents of the total TPDU processing time in the TCP/IP case, and for only eight percents in the VMTP case. All other entries reflect protocol-independent processing times. First, the measured reduction of the use of the processing resource for driving the protocol itself illustrates that VMTP is really a lightweight transport protocol. But these measurements also explains the failing of the lightweight transport protocol

TCP/IP	
Component	Processing Time
User-kernel move	13%
Checksum	12%
Device DMA	20%
TCP/IP	16%
Others	30%
Unaccounted	8%

VMTP	
Component	Processing Time
Data movement	40%
Kernel	25%
Ethernet Driver	15%
VMTP	8%
Checksum	10%

- Figure VII.1 -
The audit of the TPDU processing time,
for TCP/IP and VMTP.

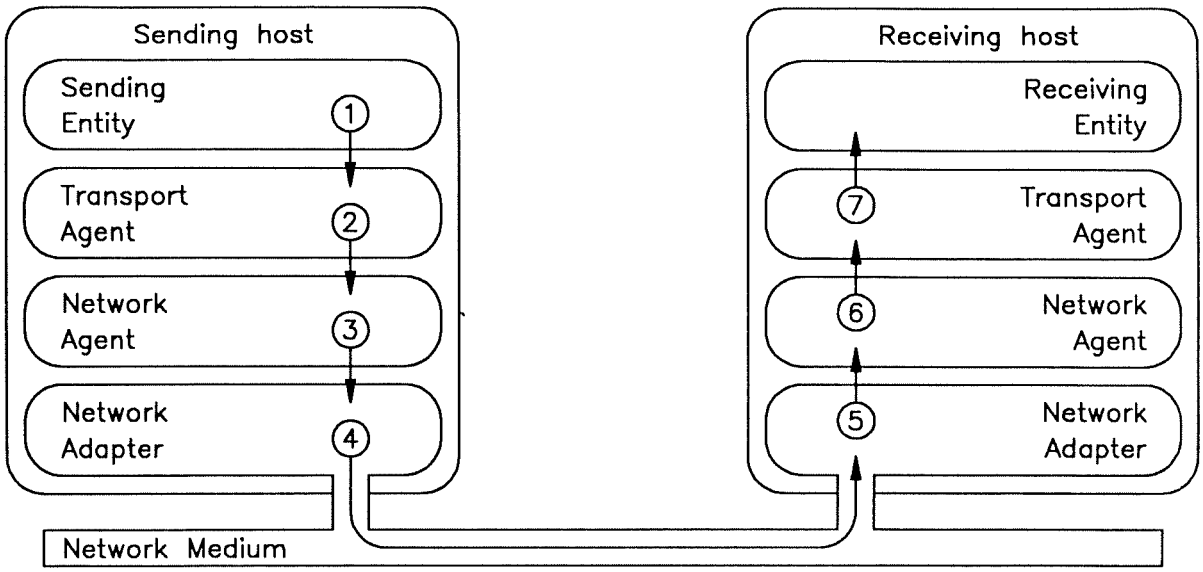
concept to achieve really high throughput: changing the protocol definition only issues less than twenty percents of the computing overhead.

The existence of the 'checksum' entry is inherent to the sequential aspect of the processing. On TPDU emission, the entire TPDU must be build, then its checksum must be computed, then it must be moved to the network layer. On TPDU reception, the entire TPDU has to be moved from the network layer, then its checksum must be verified, then it can be processed. With a possibility of parallelism, this entry can be suppressed, with the checksum being computed while the transfer to or from the network layer takes place.

In fact, the two main reasons of computing resource consumption are:

- the data movements, for thirty-five to forty percents of the total consumption,
- the operating system calls, for twenty-five to thirty percents of the total consumption.

In a very primitive implementation, the all user data has to be copied seven times to get from the source entity memory space to the destinating entity memory space. This is depicted in figure VII.2. The first copy occurs when the transport agent of the sending entity builds the DATA-TPDUs it has to transmit: the entire user message is progressively copied in the Data field of these DATA-TPDUs. The second copy occurs when the network agent of the sending host builds the datagrams it has to route: the DATA-TPDUs are copied into the body of these datagram. The third copy is caused by the network adapter, which implements on hardware the lower layers of the communication stack: the network agent of the sending host has to move all the datagrams it has to transmit into this network adapter. The fourth copy



- Figure VII.2 -
The seven copies of the user data
in a dumb implementation of the transport agents.

is the transmission of the frames including the datagrams from the network adapter of the sending host to the one of the destinating host. The fifth copy is the one of all datagrams received by the network adapter of the destinating host to the memory space of the network agent of that same host. The sixth copy is the one of the TPDUs extracted from the received datagrams, from the network agent to the transport agent of the destinating host. The seventh and last copy is the one of the Data field of these TPDUs to the memory space of the destinating entity.

Better implementations of this communication stack allow only five copies to transfer the user messages from the originating entity memory space to the destinating entity memory space. They avoid all unneeded copies of data inside the host memory using the 'chained buffers' techniques. But five copies are a minimum when working with network adapters only implementing the physical and link layers:

- First, the datagrams to transmit over the network medium must be entirely built before transferring them to the network adapter: this incurs at least a copy of the user data inside the sending host memory.
- Second, these datagrams must be transferred from the sending host memory to the sending host network

adapter, using some DMA[1] facility, or block transfer bus protocol.
- Third, these datagrams must be transferred in frames, from the network adapter of the sending host to the one of the destinating host.
- Fourth, these datagrams must be transferred from the receiving network adapter to the destinating host memory.
- Fifth, the user data part of the DATA-TPDU contained in these datagrams must be copied to the destinating entity memory space: this leads to at least one copy of the user data inside the destinating host memory.

The important overhead incurred by the operating system functions can easily be explained. Some such functions are executed due to explicit calls from the software implementing the communication stack, for the use of timers for example. The problem is: the operating system is general-purpose. And so these functions are undoubtedly not optimized for their utilization by a communication stack. But this not justifies such a computing overhead.

The computing overhead due to the operating system mainly comes from its implicit use. Several functionalities of the communicating stack must be implemented using processes that sleep until a given event occurs. Here are some examples:
- a process responsible for the processing of incoming datagrams must sleep until the network adapter awakes it for the delivery of a new datagram,
- a process responsible for the retransmission on timeout of a TPDU must sleep until its associated timer expires,
- a process responsible for a rate controlled transmission of TPDUs must sleep between burst transmissions, waiting its associated timer to expire.

The computing overhead problem comes from the fact that another process is running at the time a process is awakened. Thus the operating system must execute a 'task-switching' to allow the process to react to the event that awakened it. The problem is: a task-switching operation is very time consuming[2]. It is fair that the cost of most of these task-switchings is distributed over several TPDUs. But unfortunately, at least one context-switching is necessary for each transmitted TPDU: the one awakening the process responsible of the processing of received TPDUs on the host of the receiving entity.

---

1. Direct Memory Access, which allows a device connected to the host bus (a network adapter, for example) to access the host memory concurrently with the host central processor.
2. At least in regard to the time needed to realize the protocol-related processing of a TPDU.

*** The smart network adapters principle[3].**

To summarize the preceding analysis of the TPDU processing time audit, the main overhead associated to a TPDU transmission comes from:

- a data copy inside the host memory of the sending entity,
- a task-switching inside the host of the receiving entity,
- a data copy inside the host memory of the receiving entity.

These three overhead components are in fact all related to a single functionality of the transport protocols: the packetization of user messages into several TPDUs. And their existence comes from the fact that a host communicates with its network adapter on the TPDU basis or, for the purpose to be more accurate, on the datagram basis. If a host were communicating with its network adapter on the transport user message basis, all these overheads would immediately vanquish.

But such a network adapter would have to be far more intelligent than the normal ones. If it realizes the packetization of the user messages, it has also to realize all the computing that comes beyond this packetization:

- It has to build the actual TPDUs, including their header and their checksum.
- It has to encapsulate these TPDUs into correctly presented datagrams.
- It has also to deliver these datagrams to the link layer part of the network adapter, with the correct destinating link level address.

Furthermore, a transport agent is also designed to handle several simultaneous connections, so it may sometimes have to transmit several user messages from several different connections at the same moment. It would be totally unthinkable to enforce a sequential transmission of all these user messages. And thus it would be required for a network adapter realizing the user messages packetization to allow the simultaneous transmission of several user messages. The conclusion is: such a network adapter would also have to ensure the scheduling of datagram transmissions. This scheduling can be very complex for transport protocols using rate-based flow-control.

This kind of network adapters, which:

- packetize themselves the user messages,
- realize some additional computing relative to the transport and network protocols,
- realize some additional computing relative to the scheduling of the datagrams transmission,

are referred to as the 'smart network adapters'.

---

[3]. The justification of the need of these smart network adapters may be read in [NAB].

Beyond the suppression of the computing overhead associated to the user message packetization by the host, the use of smart network adapters lead to many other enhancements.

First, the additional functionalities these smart network adapters must ensure are no more provided by the host itself: some parallelization is incorporated to the protocol computing. This can clearly offer some performance improvements.

Second, the adapter itself is totally dedicated to its task. Thus its design can be optimized in that direction. For example, it is up to this adapter to compute the checksum protecting the integrity of each transmitted TPDU. It has already been observed that this operation can be parallelized with the operation of delivering the datagram encapsulating such a TPDU to the link layer part of the adapter.

Third, for a transport protocol defining some rate-based flow control, a smart network adapter can implement hardware-driven timers with the appropriate resolution. This solution thus allows a transmission rate control at the TPDU level, instead of at the burst of TPDUs level, as it had to be done in the software implementations of the transport agents.

This solution of using smart network adapters is a brand new one. And so there is not yet any stable solution in that matter. Two entirely dissimilar approaches are currently being studied, with VMTP and XTP. There are presented as case studies in the two next paragraphs.

VII.3. A first case study: the NAB for the VMTP protocol.

The first interesting smart network adapter to study is the Network Adapter Board (NAB). It has been designed to implement the VMTP protocol for the VMP computer, a multiprocessor to be operated using the V system[4].

The strategical decisions for the design of this smart network adapter are the following ones:

- the NAB has only to ensure the packetization and the transmission of the packets groups defined by the VMTP protocol[5]. The remaining aspects of VMTP are to be implemented in the host itself.
- the NAB has to operate a traditional processor to realize most its operations: there must only be minimal hardware developments for the realization of this board.

---

4. A comprehensive description of the NAB can be read in [NAB].
5. The concept of packets group used by VMTP has been defined in paragraph IV.3, under the title "VMTP and the improved design". Its definition can also be read in [RFC-1045].

The motivation of the first decision is that the NAB device is to be shared by several processors. If too many functionalities were defined for that board, there would be a risk that it becomes the bottle-neck of the system. The result of the second decision is a fast and easy development of the board, as well as an important aspect of adaptability: a NAB for an other transport protocol, or a NAB using an other LAN type, may be quickly developed, based on an existing NAB.

To minimize the work performed by the NAB, the template header principle is used[6]. The part of the VMTP transport agent which is performed in the host computes the template header to use when transmitting the packets group, as well as the link level destinating address. All these informations are transferred to the NAB with the data part of the packets group to transmit. The NAB packetizes this packets group into the appropriate VMTP packets, using the header template it was delivered. The only information the NAB has to add to this template header is the position of the Data field of the VMTP packet relative to the current packets group, and the appropriate checksum value. When a VMTP packet is ready, it is delivered to the link layer part of the NAB as well as the link level destinating address the NAB was given.

All the logic associated to the handling of the state machinery of a VMTP transaction is coded in software performed by the host itself. The situation is also identical for the VMTP managers: they are also implemented in the host itself. In fact, the NAB can be logically viewed as a module defining two functions of the implementation of VMTP: 'SendPacketGroup()' and 'ReceivePacketGroup()'. The physical operating of this NAB is realized using the Universal Network Device Interface Protocol (UNDIP)[7].

With UNDIP, the logical use of the function 'SendpacketGroup()' is implemented by the transmission of a structured message to the NAB. This message is referred to as a Transmission Authorization Record or TAR. Its main fields are:

- the template packet header to be used,
- a description of the data to transmit in this packet group, roughly: a pointer to this data,
- an interrupt mask.

The semantic of two first mentioned fields is straightforward, from the description of the functionality of the NAB. The interrupt mask commands the NAB when to interrupt the host when sending the packet group. By requesting an interruption at the end of the packets group transmission, the transport agent can set up its timer for the retransmission on timeout of the entire packet group, for example.

---

6. This concept has been discussed in paragraph VI.2, under the title "template header caching".
7. A comprehensive description of this interface protocol can be read in [UNDIP].

With UNDIP, the logical use of the function 'ReceivepacketGroup()' is also implemented by the transmission of a structured message to the NAB. This message is referred to as a Reception Authorization Record or RAR. Its main fields are:

- the template packet header of the TPDUs that belong to this packets group,
- a description of the buffer where to store the data part of this packet group, roughly: a pointer to this buffer,
- an interrupt mask.

The semantic of the first two fields is quite clear: they describe which TPDUs to receive and where to store their data part. The interrupt mask commands the NAB when to interrupt the host during the reception of the described packets group. For example, a VMTP transport agent can expect an interruption on reception of the first TPDU of the described packets group and another one on reception of the last TPDU of this same packets group. The first interruption allows the VMTP transport agent to set up a timer expiring when there is no more chance that subsequent TPDUs of this packets group will be received. The second one allows the VMTP transport agent to cancel the preceding timer, and to process the now entirely received packets group. If this timer expires before the second interruption, the VMTP transport agent can cancel this RAR and start the retransmission on demand procedure for that packets group.

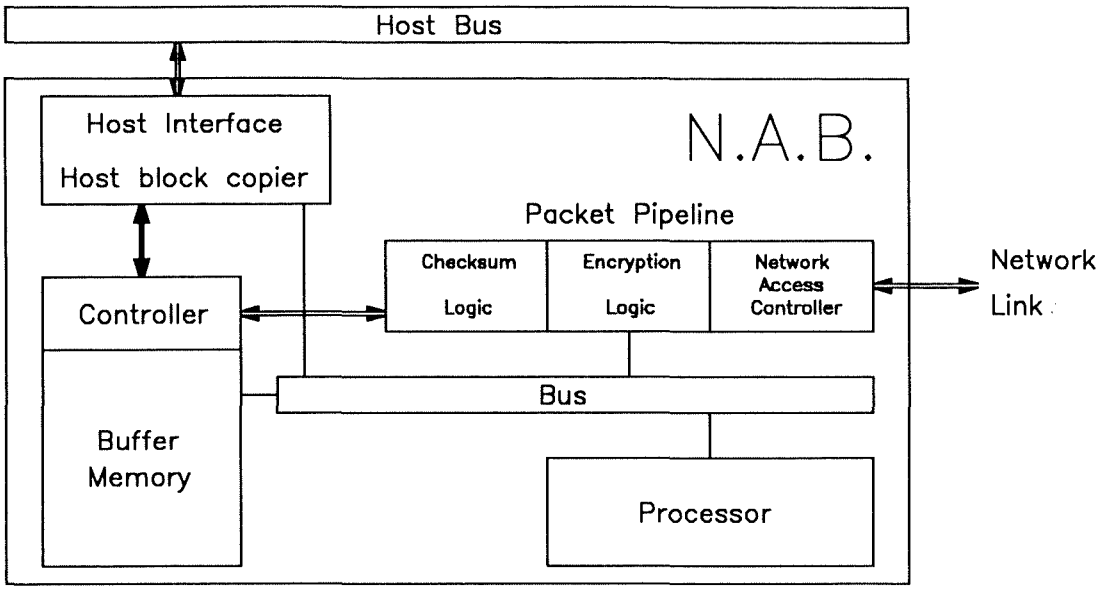
The use of these RARs and other UNDIP structured messages provides a supplementary functionality to the NAB: it can play the role of a firewall between the network and its host. In fact, the NAB only delivers its host the packets groups it was authorized to. The garbage TPDUs are dropped by the NAB, and so are also the TPDUs originating from unauthorized entities. In a traditional software implementation of a transport agent, the host must process all these TPDUs for taking the decision to drop them. And if the rate of reception of these unwelcome TPDUs were becoming too high, the transport agent would use all the available computing resource to reject these TPDUs: the normal activity of the host would then be totally frozen.

The architecture of the resulting board is depicted in figure VII.3[8]. Its five main components are:

- the adapter bus,
- the central processor,
- the buffer memory,
- the host interface with its host block copier,
- the packet pipeline.

The packet pipeline generates and checks the VMTP-level checksum. It also performs encryption and decryption[9] of the

8. This figure and its explanation come from [NAB].
9. The reader is pleased to refer to [RFC-1045] to read the explanation of the existence of this functionality at the transport level.



- Figure VII.3 -
The architecture of the NAB.

VMTP packets. These two operations are realized 'on-the-fly', while transferring the VMTP packets to or from the network. The host interface ensures the communication between the host memory and the buffer memory of the NAB. The communication between the host memory and the host interface is realized by the host block copier, using a burst-transfer bus protocol when such a protocol exists for the host bus. The buffer memory uses Video-RAM chips, which allow fast random access (60 ns), and to very fast sequential access (40 ns). This sequential access is used to exchange blocks of data with the host interface and the packet pipeline[10]. Finally, the on-board processor manages all the above devices and processes some functions associated to the protocol.

VII.4. A second case study: the Protocol Engine of XTP.

The second smart network adapter to be presented in this chapter is the Protocol Engine (PE) for the protocol XTP. This smart network adapter is to be built by a society named Silicon Graphics, Incorporated. In fact, this society is also the proprietary of the transport protocol XTP[11].

[10]. So, the number of copies of the user data was already lowered by the use of a smart network adapter, and furthermore the NAB ensures that the remaining copies are realized using the fastest existing means.
[11]. An overview of the XTP/PE project can be read in [XTP-3].

The strategic decisions for the realization of XTP/PE are totally different from the ones that lead to the design of the NAB:

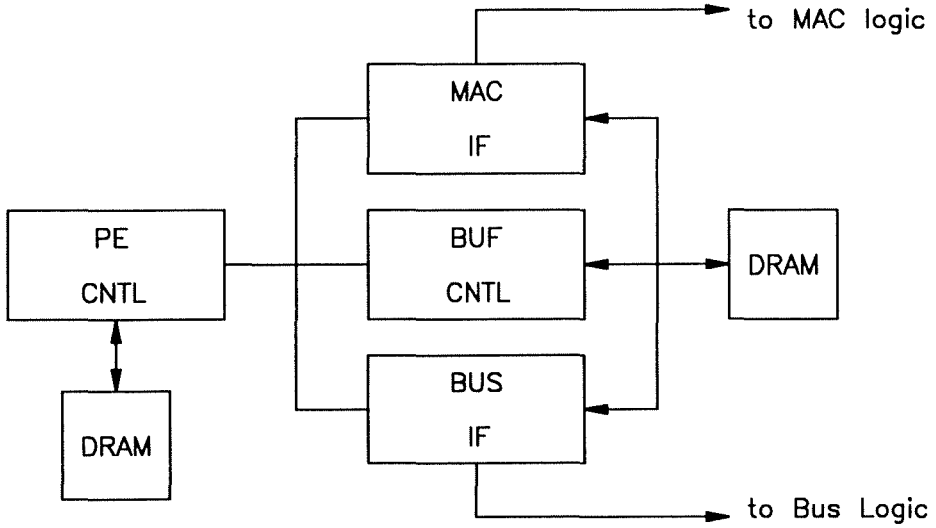
- First, the XTP/PE must implement all functional aspects of XTP, in a way to entirely free the communicating hosts of transport protocol handling considerations.
- Second, The XTP/PE is not to be architectured as a special-purpose computer, but as a VLSI[12] chip set.

The justification of these two decisions is the requirement to allow the streaming of data at the raw network media rate. This requirement can only be met if the processing of a received packet can be achieved within the arrival time of the latter. The creators of the XTP/PE think this is only possible with the protocol entirely hard-coded in a VLSI chip set.

The initial performance goal of this smart network adapter is in fact a throughput of a hundred megabits per second, which corresponds to the bandwidth of the FDDI optic fibre network. But the architecture of the XTP/PE has been chosen specifically for the ability to scale up to the one gigabit per second level.

The general architecture of the XTP/PE smart network adapter is depicted in figure VII.4[13]. The MAC IF circuit develops an interface between the protocol engine subsystem and a MAC implementation. The BUS IF circuit is an interface between the protocol engine subsystem and the bus of its host. The BUF CNTL

- Figure VII.4 -
The architecture of the XTP/PE.



12. Very Large Scale of Integration.
13. This figure and the corresponding explanation come from [XTP-3].

circuit manages the data buffers, residing on DRAMs. To realize the processing of the packet within its arrival time, this circuit will have to access the DRAMs with a bandwidth three or four times that of the MAC IF circuit. The last circuit, PE CNTL, controls all the other ones, and contains the actual implementation of XTP.

VII.5. The results of the smart network adapters.

There is not yet any operational implementation of any smart network adapters, as the idea is a very recent one. Nevertheless, very detailed simulations were realized for the NAB[14].

A first interesting simulation of the NAB concerns the use of a MC68020 as the on-board processor. This is a widespread processor, providing a computing capacity of two mips[15]. The results show that an end-to-end throughput of about forty-five megabits per second can be expected. They also indicate that the rate of exchange of packets between two such NABs can reach ninety megabits per second.

A second simulation supposed the NAB uses a AMD29000 as the on-board processor. The latter is less commonly used than the MC68020, but is far much faster, providing seventy mips as computing capacity. The results show that two NABs can then exchange packets with a rate of four hundreds and eight megabits per second.

VII.6. Analysis of the smart network adapter concept.

The results of the simulation of the NAB smart network adapter show that a very important improvement in the end-to-end throughput. As a matter of fact, a software implementation of VMTP allows a throughput ranging from two to four and a half megabits per second[16]. And the NAB implementing VMTP with a widespread processor could offer a throughput that reaches forty-five megabits per second. This is an improvement by a factor of ten.

So the smart network adapters seem to be the ideal solution to provide high throughputs for the transport protocol implementations. And the next question is: do all transport protocol fit an implementation using a smart network adapter

14. The results of these simulations were presented in [SLIDE], by Dr. Kanakia.
15. Million of Instructions Per Second.
16. This has been discussed in the paragraph V.4, under the title "the results of VMTP".

such as the NAB? Or, to be more accurate: can a NAB be designed for the classical transport protocol TCP? The response seems to be negative, mainly for two reasons.

The first reason comes from the checksum computing. A non-negligible source of performance improvement, in the NAB, comes from some parallelization computing, expressed by the use of the packet pipeline. The latter generates and checks the checksum while it is transferring the packet to or from the network. But this pipeline can only be used for a protocol with a trailing checksum, and TCP has its checksum stored in the header.

The second reason is a more definitive one. TCP is not specified to transmit a stream of user messages from an entity to another one, but a simple stream of bytes. Nevertheless, the entities using TCP logically exchange a stream of messages. In a software implementation of TCP, there is no problem with this situation:

- TCP collects in a buffer the data to be transmitted for a given connection,
- when a sufficient amount of data is present in this buffer, it is actually transmitted[17],
- TCP also offers a service to its users, enforcing the transmission of all the data present in the above buffer[18].

So, for a user to transmit its logical stream of messages, it delivers this stream of messages to TCP, but enforce each message transmission just after its delivery to the transport agent. It must be observed that the mechanism described above only exists for implementation purpose: the TCP carrying the last part of the data that have been 'pushed' has no identifying mark.

But a smart network adapter has to interact with its host using data aggregates longer than the datagram, and smaller than the entire stream exchanged in a TCP connection. There is no problem on the sending end, as the above principle can also be used. But the real problem comes with the smart network adapter of the host of the destinating entity. It has no base to justify a decision to transmit to its host the data it has received so far for a given connection. For instance, after receiving a TCP packet, a smart network adapter must face the following dilemma:

- it has to wait for the reception of the next TCP packet, in a way to have more data to transmit to its host in a single interaction,
- it has to immediately transmit the received data to its host, as the reception of the next TCP packet

17. If this data is inside the transmission window, of course. This has been discussed in paragraph III.2, under the title "the transmission window".
18. This service is sometimes referred to as the 'push()' function.

may depend on the processing of the data contained in the current one[19].

And in fact, from the second consideration, the smart network adapter has to transmit the data it has received after each TCP packet reception. It is therefore of no use at all.

To benefit from an implementation using a smart network adaptor, the first and only requirement for a transport protocol is that it must define data aggregates which are bigger than the network datagram, but shorter than the entire connection content. This can be achieved by using the notion of user message, as in the XTP case, or the notion of packets group, as in the VMTP case. For the implementation of this smart network adaptor to be very performing, the layout of the exchanged TPDUs must be chosen with a great care. For example, the checksum must be the trailing information of each TPDU.

VII.7. Conclusion.

It is possible to design really fast implementations of a transport protocol. The solution is simply to move the packetization functionality onto a smart network adapter.

The requirements for a transport protocol to allow such an implementation are very few, and really not constraining. But the situation is that the currently most used transport protocol, that is to say TCP, does not meet these basic requirements.

19. This is the case when the currently received TCP packet contains the last part of a user message, which must be responded by the destinating entity before the delivery of next user message, so before the next TCP packet can be received.

Conclusion.

The first requirement of a transport protocol is to allow a reliable transmission of data. As this transport protocol is implemented over a network layer only providing a minimal functionality, it has to include mechanisms detecting and correcting the inherent annoying problems of such a network layer.

Even when these mechanisms are chosen in a way to use the network layer with a satisfying efficiency, the end-to-end throughput allowed by the transport layer stays far away from the one of the network layer. This lead to the conclusion: the transport protocols are slow because they consume too much computing resource.

Based on this observations, two solutions were proposed, both expecting a radical slash down of the TPDU time processing.

The first solution dealt with redefining the transport protocols themselves. It is the lightweight transport protocol concept, which consists of specifying a transport protocol for it to 'naturally' lead to a fast implementation. The implementations of these protocols showed an improvement at the throughput point of view, but not so radical as expected.

The second solution dealt with focussing on the implementation of the transport protocols without changing their definition. It consisted on finding the most appropriate strategy to implement each functionality the transport agents perform. These optimized implementations resulted to better improvements than the lightweight transport protocols concept.

The comparison of the two preceding solutions indicated that two classes of functionalities can be separated in the transport agents computing: protocol-specific operations and protocol-independent operations. And the better results of the second solution prove that it is more interesting to improve the computing of this second class of functionalities.

An audit of the TPDU processing time allowed the discovery of these very time consuming protocol-independent operations. The importance of the data movements and the operating system use were highlighted. The responsibility of these computing overheads was associated to the interaction of the host with its network adapter on the datagram basis. These considerations lead to the expression of the real solution of the end-to-end throughput: the smart networks adapters.

This solution consisted of moving a non-negligible part of the functionalities of the protocol handling outside the host, on the network adapter. This fact of moving these functionalities onto a special-purpose board authorized the design of very fast hardware-assisted implementations.

Then the question of the influence of these solutions on the transport protocols specification was answered: there are very little requirements to allow an implementation using a smart network adapter. The only ones are in fact:

- the need to transmit a stream of user messages,
- the need to have a trailing checksum field in the transmitted TPDUs.

At this point, it was also demonstrated that the currently most widespread transport protocol, TCP, can not benefit from an implementation using a smart network adapter, as it controls the transmission of a raw stream of bytes.

A last interesting question to raise about this analysis of the performance of the transport protocols concerns the possible future trends.

It is clear that a project such as the XTP/PE totally answers the end-to-end throughput question: it ensures the streaming of data at the raw network bandwidth. Likewise, the simulations of the NAB show that its performance is in direct relation with its computing capacity. So, the processor to use on a NAB can be chosen in respect to the expected end-to-end throughput.

So it is now time to wait for the response of the computer scientists to the availability of these very high end-to-end throughputs. Their new wishes will show the right direction for the subsequent researches in that matter.

One of the most plausible evolution is a growing use of the new paradigm of the computer science: the widely distributed systems and the client/server model. If this happens, the concept of 'performance of a transport protocol' will also evolve. Beside the need of high throughput for the transfer of large user messages, there will be the requirement to allow the transfer of short user messages with a very low latency.

The problem is the contradiction between these two aspects. So the transport protocols are worth another analysis with this new idea in mind...

Appendix I:

References.

[ISO-8073] "Connection Oriented Transport Protocol Specification", International Organization for Standardization, ISO 8073.

[ISO-8473] "Connection-less-mode Network Service Protocol Specification", International Organization for Standardization, ISO 8473.

[NAB] "The VMP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors" by H. KANAKIA and D.R. CHERITON, proc. SIGCOMM'88 Symposium on Communications architectures and Protocols, 175-187.

[NETBLT] "NETBLT: A High Throughput Transport Protocol" by D.D. CLARK, M.L. LAMBERT and L. ZHANG, ACM SIGCOMM'87 Workshop on Frontiers in Computer Communications technology, 353-359.

[PERFORMANCE] "Measured Performance of Transport Service in LANs" by L. SVOBODOVA, Computer Networks and ISDN systems n°18, 31-45.

[TCP-1] "An Analysis Of TCP Processing Overhead" by D.D. CLARK, J. ROMKEY and H. SALWEN, proceedings of the 13th conference on local area networks, 284-291.

[TCP-2] "Congestion Avoidance and Control" by V. JACOBSON, Proc. ACM SIGCOMM'88 Symposium on Communications architectures and Protocols, 314-329.

[RFC-768] "The User Datagram Protocol" by J. POSTEL, Network Information Center, RFC-768.

[RFC-783] "The TFTP Protocol" by K. SOLLINS, Network Information Center, RFC-783.

[RFC-791] "Internet Protocol" by J.B. POSTEL, Network Information Center, RFC-791.

[RFC-793] "Transmission Control Protocol" by J.B. POSTEL, Network Information Center, RFC-793.

[RFC-998] "NETBLT: A Bulk Data Transfer Protocol" by D.D. CLARK, M.L. LAMBERT and L. ZHANG, Network Information Center, RFC-998.

[RFC-1030] "On Testing the NETBLT Protocol over Divers Networks" by M.L. LAMBERT, Network Information Center, RFC-1030.

[RFC-1045] "VMTP: Versatile Message Transaction Protocol, Protocol Specification" by D. CHERITON, Network Information Center, RFC-1045.

[RPC] "Exploiting Recursion To Simplify RPC Communication Architectures" by D. CHERITON, Proc. ACM SIGCOMM'88 Symposium on Communications architectures and Protocols, 76-87.

[SLIDES] Slides of the conference "New Protocols and High Speed Networks" by H. KANAKIA, held from 16 to 18 of January 1990 at the "Univertsité libre de Bruxelles" and organized by the "Chaire IBM 1989-1990".

[UNDIP] "Universal Network Device Interface Protocol (UNDIP)" by H. KANAKIA and D. CHERITON, proceedings of the 13th conference on local area networks, 301-309.

[XTP-1] "Express Transfer Protocol Definition, Rev.3.2." by G. CHESSON, Protocol Engine Incorporated, June 1988.

[XTP-2] "A Lightweight Transfert Protocol For The U.S. Navy Safenet Local Area Network Standard" by M. COHN, proceedings of the 13th conference on local area networks, 151-156.

[XTP-3] "XTP/PE Overview" by G. CHESSON, proceedings of the 13th conference on local area networks, 292-296.

Appendix II:

Index.